Project EC7: Embeddability of IA components

Action Sheet 12

contact@confiance-ai.fr | www.confiance.ai

Specific information to be provided in ANR reports only

Progress Report		Final Report	
From:	*AAAA/MM/JJ*	To:	*AAAA/MM/JJ*
Brief description of the project	********		
Contractual description of the project:		********	
Reference budget:		********	

Document reference: XXX

Contributors

	Name	Organisation	Role
Responsible for the deliverable	MAXIM Cristian	IRT System X	EC7 Contributor
Responsible for the deliverable	GABREAU Christophe	Airbus	EC7 Contributor
Co-author			
Co-author			

Document Control

Revision	Date	Commentary	Author
V0.0	10/01/2022	document creation	
V0.1	10/02/2023	document delivery	
V0.2			
V1.0			

Contents

Chapter A

Guidelines for software/hardware implementation of certifiable AI component

A.1. State of the art

In general, certification constraints refer to the formalisation of constraints that the organisation, the development process or the product itself shall satisfy for the safety of the system to be guaranteed. This includes in particular all constraints already stated in the previous paragraphs about hardware resource usage and timeliness. The main difference lies in the fact that (i) those constraints are formally stated into some applicable regulatory documents (e.g., ED-12C/DO-178C or ED-80/DO-254, ISO 26262, ECSSs, etc.), (ii) some specific evidences are required to demonstrate that those constraints are satisfied (mainly the outcomes of the process activities e.g. test cases review report) , and (iii) , in some cases, some means are required to achieve (ii) and (iv) satisfaction of constraints (i) to (iii) is assessed by a third-party and independent body.

Therefore, in the context of EC7, identifying the pertinent certification constraint boils down to identifying constraints related to (i) resource usage, (ii) time determinism , (iii) fault tolerance and (iv) safety properties preservation expressed in the standards.

In this chapter we will present the ML certification efforts by domain. As an example, the aeronautics domain has a well established certification procedure and its adaptation to ML has already started in the new EUROCAE (European Organisation for Civil Aviation Equipment) WG-114 workgroup. Similarly, the automotive industry tackles this subject in the ISO 21448 "Safety of the Intended Functionality" (SOTIF) standard.

While in the aeronautics and automotive industries the subject of ML certification is being treated, in other industries the efforts stay minimal due to the lack of a certification authority or the fact that usage of ML is not yet very spread. Nevertheless, in this chapter we mention any paper that aims at a general framework for ML certification through a certain number of methods, considerations or steps.

A recent survey of exiting papers dealing with the certification of Machine Learning is pre-

sented in [Tambon et al., 2021]. This paper separates the indexed references according to the criteria that they try to approach/prove in the certification process. The identified criteria are: robustness, uncertainty, explainability, verification, safe reinforcement learning and direct certification. For our state of the art, the approach is different and we are separating the existing publication by domain and identify the criteria of importance for each domain.

Jenn et al. [Jenn et al., 2020] present the main challenges on the certification of systems embedding ML. This is a resuming paper of the Dependable and Explainable Learning (DEEL) Certification Workgroup [Delseny et al., 2021].

Some emerging standards and guidance, have been proposed to better reflect the ML lifecycle, e.g., the us Federal Drug Administration (FDA) proposed regulatory framework on AI/ML-based Software as a Medical Device (SaMD) [Food et al., 2019], UL4600 [Koopman et al., 2019] and Assurance of Machine Learning for Autonomous Systems (AMLAS) [Hawkins et al., 2021]. However these approaches are not yet mature and further work has to be done in each domain for clear and precise certification standards to be produced. This is one of the motivation of this document in which we are contributing to collective effort of introducing AI in embedded critical systems with a increased level of confidence.

A.1.1 Automotive domain

In the automotive domain, the main recommendations about *functional safety*[1] are gathered in the ISO 26262 series of standards (“Road vehicles - Functional Safety"), a declination of the IEC 61508 addressing the specific needs of electrical and/or electronic systems embedded within road vehicles.

The ISO 26262 gives procedures and measures to prevent and handle failures resulting from random hardware faults or systematic HW/SW faults. It is complemented by the ISO 21448 “Safety of the Intended Functionality” to address *hazardous behaviour caused by the intended functionality or performance limitation of a system that is free from the faults addressed in the ISO 26262 series*. Quoting the standard, these limitations include:

- *The inability of the function to correctly comprehend the situation and operate safely [...]*
- *Insufficient robustness of the function with respect to sensor input variations or diverse environmental conditions*

Embarcability constraints and ISO 26262 The structure of the ISO 26262 standard is recalled on Figure A.1. For the most part, concerns about embarcability are essentially addressed in Part 5 (Product development at hardware level) and Part 6 (Product development at software level). Nevertheless, other aspects related, for instance, to the system dependability (e.g. redundancy and monitoring) or to operational monitoring and maintainability (e.g., need for instrumentation) are addressed in other parts of the standard (in particular Part 4 “Product development at the system level”). Part 11 shall also be analysed or it covers many aspects related to the hardware technologies used to implement ML components.

[1]Functional safety is defined as the *absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems*

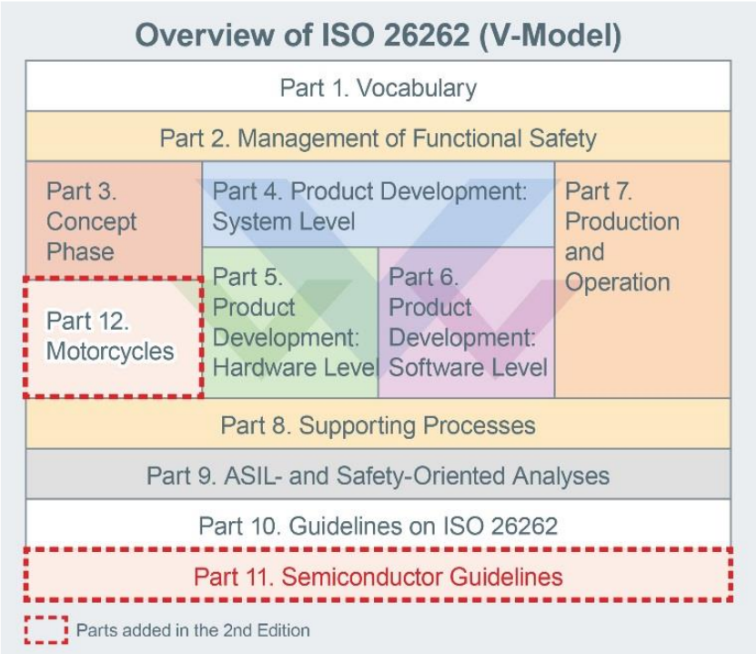

Figure A.1: Structure of the ISO 26262 (taken from Rohm's white paper at https://fscdn.rohm.com/en/products/databook/white_paper/iso26262_wp-e.pdf)

A detailed analysis will be provided in the final version of the document, *if necessary*. In principle, the analysis should be carried out in a systematic manner, requirement per requirement, measure per measure in order to identify those satisfying the criteria stated before.

Hereafter, we give a very preliminary and partial analysis of some relevant parts of the ISO 26262.

- Part 4: Product development at the system level
  - (§6.4.6) (Concerns: how to ensure partitioning when GPUs are used by several components with different ASILs)
- Part 5: Product development at hardware level
  - (§7.4.4) Verification of hardware design
  - (§9) "Evaluation of safety goal violations due to random hardware failures" (concerns: what is the impact of random hardware faults (e.g. bit flip due to radiation or EMI) on the behaviour of the neural network? how to monitor the large data areas used by models?)
  - etc.
- Part 6: Product development at software level

  - (globally) (Concerns: integration of software developed by third parties and provided as libraries, integration of tool-generated code, compliance to coding standards, testability, etc.)

- Part 11: Semiconductor guidelines
  - (§4.5.5) 'Integration of black-box IPs (concerns: black box ML acceleration IPs)
  - (§5.1) Digital components and memories (concerns: failure modes of GPUs, techniques to detect and recover errors in very large memory areas)
  - (§5.1.9): Techniques or measures to detect or avoid systematic failures during design of a digital component
  - (§5.4) Multi-core components (concerns: freedom from interferences, timing faults)

Embarcability constraints and ISO 21448 *The ISO 21448 (SOTIF - Safety Of The Intended Function) does not directly induce specific requirement on embarcability.*

Silicon Reliability and ISO26262

According to the ISO26262 standard, the *Safety Integrity Level* of each system must be evaluated based on assessing three criteria : (i) S=*Severity* if the system fails, (ii) C=*Controlability* of the driver's ability to control the situation if the system fails and (iii) the E=*Exposure* or the probability of the event to occur. Based on these criteria, systems are classified into ASIL levels from A to D where D is the most stringent. In table A.1 the ASIL classification of some common automotive applications are shown.

Table A.1: Example of ASIC Levels in ISO26262

System	ASIL Level
ADAS	B-D
Airbags	D
Cruise Control	C-D
Drowsiness Monitoring	A-B
Power Door	A-B
Steering	D / D+

Depending on the ASIL level of the system, the electronic components must meet certain quantitative targets in terms of their ability to detect faults and in terms of their reliability. The common unit for reliability is the *FIT* which stands for Failures in Time. 1 FIT corresponds to one event (failure) every 10^9 hours.

Designing integrated circuits with a failure rate as low as 10 FIT is very difficult. The failure rate of an integrated circuit is roughly proportional to its area. Current AI accelerators, such

Table A.2: ISO26262 Fault Detection and Reliability Metrics for Different ASIL Levels

ASIL	**B**	**C**	**D**
Single Point Fault Detection	$\geq 90\%$	$\geq 97\%$	$\geq 99\%$
Latent Faults	$\geq 60\%$	$\geq 80\%$	$\geq 90\%$
FIT	≤ 100	≤ 100	≤ 10

as GPUs and TPUs, have large amounts of embedded memory and multiple parallel processing units and are much larger than previous automotive micro-controllers which typically contained one (or a few) CPU(s) plus simpler peripherals. Off-the-shelf AI accelerators will clearly not meet the reliability and fault detection targets required by ASIL-C or ASIL-D. A recent integrated circuit from Renesas [Matsubara et al., 2021] has been designed to provide hardware acceleration for CNNs in automotive applications. It achieves the above hardware fault detection and reliability metrics through the use of redundancy as shown in A.2. The main ARM processors operate in dual-core lock-step (DCLS) which is typical for automotive applications, and the CNN accelerator portion can be configured as a main and a checker group.

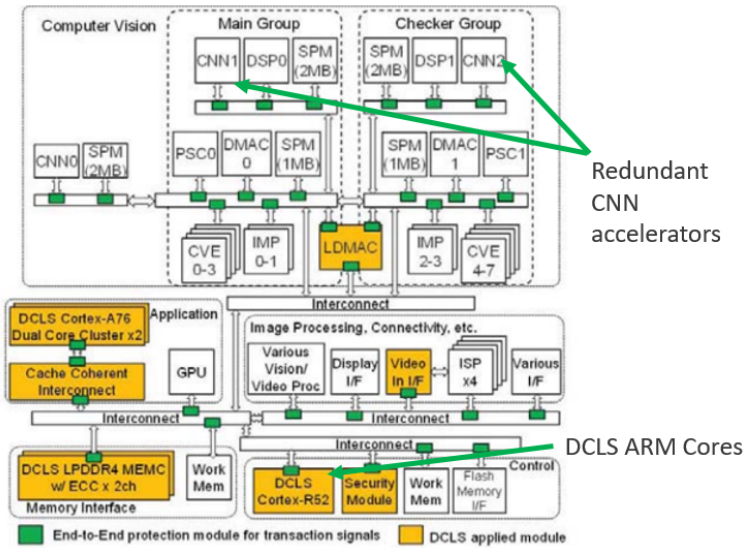

Figure A.2: Renesas 12nm Automotive Driving Processor [Matsubara et al., 2021]

Unfortunately, CNN accelerators currently consume significant power. For example, most of Google's TPUs consume over 200W. If this power consumption must be doubled to achieve redundancy, then it starts to represent a non-trivial fraction of the total power budget of an electric vehicle. For this reason, there is growing research activity underway to find alternative ways to detect and mask the impact of faults that occur in hardware used for CNN accelerators. This research is explored in a dedicated section of the state of the art.

A.1.2 Aeronautics domain

Considering the definition of the "embarcability" problem, the analysis of certification standards should normally consider all HW and SW development activities. In the aeronautics domain, this would mean considering all objectives from the ED-12C/DO-178C or ED-80/DO-254 standards. As an example, the overview of the ED-12C/DO-178C standard processes is recalled on figure A.3.

In order to anticipate future EASA guidance and requirements for safety-related machine learning (ML) applications, EASA proposed a concept paper in December 2021 [EASA, 2021]. In the search for strategies to verify CNN-based systems against hardware faults at the inference stage, EASA launched the project titled “Concepts of Design Assurance for Neural Networks” (CoDANN) [Daedalean, 2021a] and its suite [Daedalean, 2021b]. These projects examine the challenges posed by the use of neural networks in aviation, in the broader context of allowing machine learning and more generally artificial intelligence on-board aircraft for safety-critical applications.

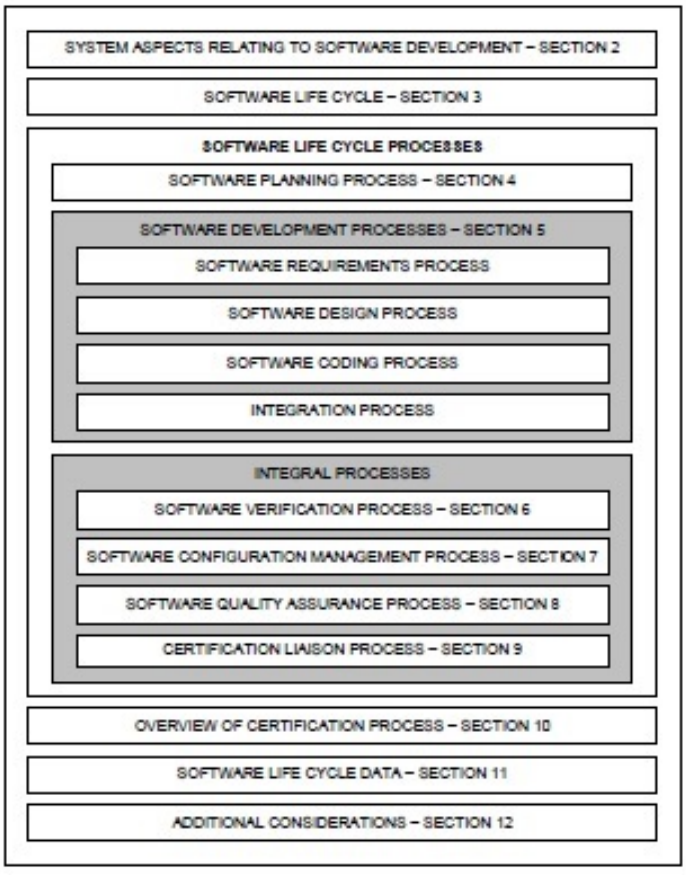

Figure A.3: ED-12C/DO-178C overview

However, in order to keep this document short and focused, we exclude from the analysis all activities for which we consider that the development of a ML model does not strongly differ from the development of any other "traditional" (i.e., non-ML) application.

So, the approach consists to focus on activities or objectives for which ML is *expected* to raise particular challenges. Some of them have already been identified in the previous sections:

- Demonstration of consistency with available platform resources, *due to the fact that*
  - Implementations of ML models are often extremely demanding in terms of hardware resources and time, so
    - They are prone to reach the limits imposed by the hardware or by the applicable requirements, leaving small “security margins”
    - They often require very high performance computing platforms for which (i) no or few return of experience is available (e.g., GPUs, many-cores, ML ASIC accelerators), (ii) architectural complexity is high (e.g. high-end processors, GPUs), (iii) no or few information concerning their design are available (e.g., GPUs, ASIC accelerators)
  - Implementations often rely on complex, opaque, and rapidly evolving transformation chains, middleware, and libraries for which hardware resource and worst case execution times estimations are difficult
- Demonstration of the preservation of the functional performance of the ML model, *due to the fact that*
  - a full description of the model semantics is not provided by the native format provided by usual design platforms (e.g. ONNX)
  - The implementation chain is complex and opaque, and rely on conversion tool (from training platform to deployment environment) and optimisations (e.g. pruning, quantization) for a better efficiency of the deployed ML model inference.
  - ML-model strongly rely on numerical computations and, to implement the model on the limited resources of the execution platform (e.g., memory, LUTs or computation capabilities such as the existence of FP units), some optimisations are needed whose effects on the implemented model are difficult to determine.
- Demonstration of robustness, *due to the fact that*
  - the operational conditions and the execution environment where the ML deployed model infers may differ from the training environment. Indeed the variability of the inputs (inside or outside the ODD) can lead to unintended behaviors of the embedded system, that should be detected and mitigated with respect to the system safety requirements.

A.1.3 Railway Domain

The main actors in the railway certification are the European Union Agency for Railways (ERA) in the EU, and the Federal Railroad Administration (FRA) in USA. One of the tasks of these authorities is to promulgate and enforce rail safety regulations. This is done by imposing a series of certification requirements which each train has to respect in order to be allowed to run.

AI is already used in various operations, such as real time traffic resource and passenger flows optimization, response to unforeseen events with projected evolution, decision-making based on predictive simulations, comparison and recommendation of solutions, etc. AI, and more specifically ML, is not meant to replace rail operators, but to help them making better and faster decisions by analyzing a much larger set of parameters than the human mind can process, with the required level of safety.

Most of these examples are for the surrounding environment of a running train, while the safety critical system embedded in the train don't use any ML techniques. The reason for this is that the tasks of such a safety critical system are either automated, completely deterministic and under a strict certification by the railway authorities, or performed by human operators and that don't integrate yet any ML techniques. Using ML for the second type of tasks could increase the overall safety of the system. By introducing ML in such task, they ultimately become automated and a certification similar to the one for the first type of tasks has to be performed.

A.1.4 Medical Domain

In the process of producing a medical device, a series of ISO standards have to be respected (e.g. ISO9001 for quality and management, ISO13484 refined for medical devices to respect quality control, traceability, process validation and risk management, ISO14001 for the environment, ISO27001 for information security, etc).

For the software development of medical devices, the reference standard is CEI/IEC 62304. This is is a harmonized standard within the EU and has been recognized as consensus standard by the FDA. This standard defines the software life cycle processes to be used for the safe development and maintenance of medical device software. CEI/IEC 62304 [Han, 2007] requires initial assignment of the applicable software safety class to each software system based on severity: e.g. class A (no injury or damage to health is possible), class B (non-serious injury is possible), and class C (death or serious injury is possible).

From our knowledge, there is no evolution or this standard that would take in account the use of ML techniques in medical devices.

In [Jia et al., 2021]; the outhors present the main chalanges of using IA in the healtchare system and exeplify the observations on a case study focusing on use of mechanical ventilation in intensive care units (ICUs)

A.2. Context of guidelines proposed

Nowadays the design of a complex aeronautic system requires a top down iterative approach from aircraft level downward: the avionic functions may be performed by systems of systems,

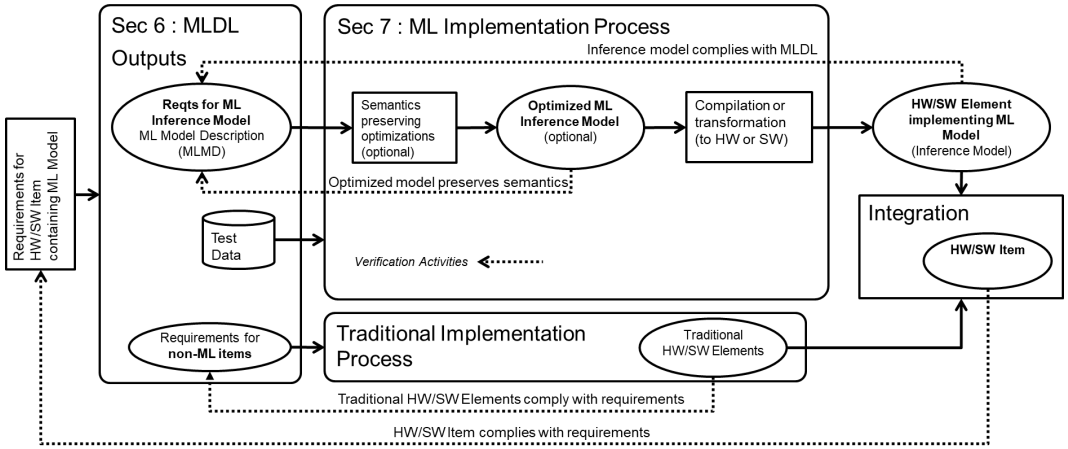

Figure A.4: WG-114- Item implementation and verification process

with systems that may be decomposed into subsystems (or equipment), that may contain a collection of software and hardware items. Therefore, avionic development considers up to 4 levels of engineering: (i) Function, (ii) System (iii) Subsystem/equipment and (iv) Item. The development process of such complex system relies on several decades of experience and good practices that keeps on being adapted today. These methods have been standardized through EUROCAE/SAE standards for system development (ED-79A/ARP4754A covering the first 3 levels of engineering) and EUROCAE/RTCA standards for software items (ED-12C/DO-178C and supplements) and hardware items (ED-80/DO-254). They are recognized as applicable means of compliance with regulation requirements by worldwide Avionic Authorities (AAs).

In the context of the study of ML embeddability constraints during implementation phase (EC7), we are going to focus on the lowest level of engineering, i.e the development of an item including a ML model. In the rest of the document, the implementation process will be based on the outcomes of the EUROCAE/SAE WG-114/G-34 work as decribed in the figure A.4 which describes the process flow of the implementation and verification of the item.

Therefore the scope of this section is the implementation & verification of the item that includes the ML model (or ML model elements) designed and validated by the previous phase. The ML model has been trained, validated and verified against the system requirements allocated to the item (as per the ARP4754A definition) it belongs to. The host item (which can be SW or HW) has been identified within the previous design architecture phase. The Item Implementation & Verification phase will focus on the transformations of the ML model from its design representation to the "*inference*" model and ultimately to its target representation (called "*deployed*" model) *without possibility of retraining*, with respect to the ML model description output from the previous phase and the host platform constraints (refer to figure A.4). The item integration and verification activities will have to demonstrate that these transformations have not altered the ML model properties and have not introduced unforeseen behaviors that could be detrimental to the safety objectives.

A.3. ML-based item implementation process

A.3.1 Item requirements capture

The objective of the requirement capture process is to specify all the characteristics, behaviours and performances of the item that contains the ML model (or the ML model elements). All the inputs from the previous phase are analysed and refined if necessary. The development of derived requirements due to low level design may be necessary to have an efficient execution of the item in its operational environment.

ML model design description - The ML model part should be fully described to such an extent that its semantics is unambiguous and does not leave any room for interpretation during implementation whatever the target. For instance for a NN: the number of layers, the number of neurons per layer, the hyper-parameters but also the internal operations used for the inference in the native environment (e.g. convolution operations implemented by the native framework such as Tensor flow or PyTorch).

Note: A certain tolerance in the semantics implementation would be acceptable depending on the safety criticality of the function. This could simplify the development and the verification of the ML model including in the item.

ML Model performance requirements - The model has been designed to meet some performance (e.g. accuracy) when inferring the test dataset in the native environment. These performance should also be attained during the inference on the target platform.

ML Model determinism requirements - The execution of the inference model (on the target platform) should respect some timing and memory constraints to correctly operate the function it implements, in the context of the item execution it belongs to.

Verification requirements - The following data sets should be provided: the test dataset used to verify the performance of the model in the design environment and any other datasets usable for verification purposes (robustness, OOD...)

Interface requirements - The interfaces with the resources of the selected target platform should be described (e.g. libraries, drivers, OS...). The user interfaces and the limitations should be known.

Monitoring requirements - Some specific functions may have to be defined to monitor and record the ML model behaviors in order to detect any degradation of the ML model performances and possible alteration of the safety requirements.

Pre/post model processing requirements - The necessary treatments before and after the model computation should be described.

Pre/post data processing requirements - The necessary processing of the input data before and after the model computation should be described.

Derived requirements - Any emerging requirements supporting some low level design constraints [*Feedback1]. For instance:

1. Architecture deployment description (allocation of models or model elements to the items identified in the susbsystem architecture)

2. Initialisations/configurations for the installation on the target

3. Numerical representation: the on-target inference may require the use of numerical representation of the weights/bias of the ML model. This constraint may alter the semantics of the design model and require a new training of the ML model [*Feedback2]

[*Feedback1]: Any derived requirements should be fed back to the system safety assessment process for evaluation

[*Feedback2]: the numerical constraints may have been defined by the implementation phase when the target is selected and pass up to the design phase as a training constraint

Traceability - Traceability from the embedded code to the model design description can be covered by a qualified tool that guarentees a trusted code generation.

Requirement standard adherence - No existing standard for the ML requirement specific part.

A.3.2 Item implementation design

The objective of the implementation design process is the transformation of the ML model (or elements of model) in an implementation model that can be hosted in the item it has been allocated to, consistently with the implementation requirements. This transformation should preserve the ML model properties and respect the operational constraints linked to the target. The activities described underneath are part of the ML model implementation design process.

Conversion - The original format of the ML model is specific to the framework on which it has been trained. A conversion is then needed to transform the model in a representation compatible to the target platform. This step is about simplifying the graph components and removing parts that are not needed to the inference in the operational environment. Whatever the changes are, they should not impact the model behaviour and performance.

Deployment constraints - Some low-level design can be necessary to efficiently adapt the execution to the host platform resources capabilities (e.g. sequence, pipeline can be very specific to optimise the execution time).

Optimisation - Other transformations can be done at this step in order to improve the efficiency of the ML model inference on the target and respect the operational constraints linked to the target (timing and memory resources). However, as no retraining is possible in the implementation phase, transformations are not supposed to change the semantics of the model. For instance we know that the Winograd algorithm optimizes the convolution computation and guarantees an identical mathematical result. The optimization techniques could be summarized as follows (not supposed to be a comprehensive list):

- Network pruning: consist in removing the neurons or weights of weak importance in order to reduce the memory footprint of the network. As this change alters the model semantics and that no retraining is authorized in the implementation phase, a re-evaluation of the performances are necessary to assess whether the degradation is acceptable with respect to the expected performances. It should be noted that the removal of neurons sounds more efficient from an implementation perspective, because the suppression of weights may lead to an irregular network that is more difficult to implement/optimize on hardware.

- Parameter quantization: aims at reducing the number of bits used to represent weights and bias. There are several quantization methods. The following are proposed as examples:

  - Weights compression: Considering there are 2 fundamental representations with different sizes: integers (8, 16, 32 bits) and floating point numbers (16, 32, 64 bits), this activity consists in converting the weights into smaller types in order to reduce the memory footprint and depending on the hardware, to allow for faster computation. The quantization degrades the numerical precision of the weights and then may alter the performance of the algorithm.
  - Weights clustering: Weights are clustered by range and averaged to a reduced number of values (i.e. if 4 values are retained then only 2 bits are necessary to index the values)

- Algebraic optimization: mathematical operations can be optimized while producing the same results. For instance, in case of a Convolutional Neural Network, Winograd optimization (reducing the number of multiplication) preserves the semantic of convolutions.

- Fusing batch normalization and convolution in runtime (Ref1): During runtime (test time, i.e., after training), the functionality of batch normalization is turned off and the approximated per-channel mean and variance are used instead. This restricted functionality can be implemented as a convolutional layer or, even better, merged with the preceding convolutional layer. This saves computational resources and simplifies the network architecture at the same time.

- Knowledge distillation: To be completed

Ref1: https://nenadmarkus.com/p/fusing-batchnorm-and-conv/

A.3.3 Item code generation and integration

The objective of the generation process is to obtain the final item (e.g. an executable for a SW item). This process transforms the ML implementation model (and associated pre/post processing) into source code and then into an object code compatible with an installation on the selected target (deployed model). Then the objects are integrated with one another to make the final item. As far as the ML model is concerned, these transformations should also preserve the ML model properties.

- SW item: Programming/Compiling/Linking - Transformation into source code, object code of each item part (including the ML model) and integration to the final item.

- HW item: Transformation of the detailed design of each item part (including the ML model) and integration to the final item.

A.4. ML-based item verification process

This process verifies that the item is compliant and robust with the item implementation requirements and more specifically that it complies and is robust with the ML model implementation requirements. This means verifying that any transformation performed on the model in between its design representation and its "deployed" representation for the targeted platform will not alter the behavior, safety and performance of the model inference.

A.4.1 Verification of the ML model

The objective is to verify that the "deployed" model correctly and completely implements the ML model description. This means verifying that any transformation performed on the model in between its design representation and its deployed representation for the targeted platform will not alter the behavior, safety and performance of the model. For the sake of the safety of the system, any deviation raised during the ML model implementation should be characterized and fed back to the safety assessment process.

ML model semantics verification - Verification that the "inference" model is compliant with the ML model description with respect to exact or approximated replication requirement. In case of an exact replication of the designed model then the "deployed" model will have to fully preserve the properties obtained during the design phase. If it is not the case, the tolerance should be properly defined and evaluated at run time to assess if the differences in the model representations are not detrimental to the functional and safety requirements of the system.

ML model performances verification - Test of the ML inference model with test dataset to verify that the ML performances has not been altered by the implementation transformations.

A.4.2 Verification of the item implementation

The objective is to verify that the item correctly and completely implements its requirements in the operational target environment.

Compatibility with the target - The objective is to ensure that no conflicts exist between the ML Model and the hardware/software features of the target platform, especially system response time and input/output hardware. The differences between the platform/environment used for training and the target implementation platform should be identified and assessed for impact on the inference model behavior and performance. This may be accomplished by traditional methods.

Non-functional performances - Verification of the non-functional requirements (related to timing and memory resources). It encompasses the correctness and consistency of the ML Model, including stack usage, memory usage, fixed point arithmetic overflow and resolution, floating-point arithmetic, resource contention and limitations, worst-case execution timing, exception handling, use of uninitialized variables, cache management, unused variables, and data corruption due to task or interrupt conflicts.

Specific implementation - Test data set may need to be augmented to cover specific implementation concerns (TBC)

A.4.3 Verification of the item robustness

The objective is to verify that the item is compliant and robust with its requirements inside the ODD (Operational Design Domain) and outside the ODD. It is also to demonstrate that any unintended behavior has been detected and mitigated according to the system safety requirements. Actually this process is iterative with the upstream system process and requires refinement of the safety requirements allocated to the item, indeed as raised by the DEEL White Paper (*): *Robustness can be tested against known or unknown conditions. When conditions are known (e.g. admissible values can be specified), particular efforts may be invested to ensure that the system will behave as intended in the presence of those abnormal conditions. When the dimension of the input space is too large to be specified (e.g. pedestrian detection), it is impossible to define exactly what an abnormal condition can be. In such conditions, robustness has to be evaluated against unknown conditions*.

With respect to the DEEL White Paper(*), the robustness of an item including an ML model is two-fold: *(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 equivalent responses within the neighbourhood of an input*. The neighbourhood of the input is refined later in the DEEL WP as small perturbations:

- Small: requires the specification of metrics, e.g. estimating the similarity between normal and abnormal (yet possible) input values. Metrics are choices of the analyst and not intrinsic properties of the world.
- Perturbations: refers to alterations of numerical quantities that may induce erroneous values. In the context of ML models, perturbations mainly affect the inputs. In addition, the implementation of the model can also be subject to numerical instabilities.

The stability is a synonym of robustness according to EASA concept paper (**) and can be considered as equivalent to the local robustness defined by DEEL. It also defines the perturbations aspects to be considered in both design and operational phases: *Anticipated MOC LM-11-1: As outlined in (Daedalean, 2020) Section 6.4.1, there are (at least) two sources of instability which should be analysed*:

- Perturbations in the design phase due to fluctuations in the training data set (e.g. replacement of data points, additive noise or labelling errors). This analysis supports the demonstration of the learning algorithm stability.
- Perturbations in the operational phase due to fluctuations in the data input and prediction output or in the model itself. This analysis supports the demonstration of the model stability.

Therefore the stability is a model property that is to be verified during the learning phase (design) but also during the implementation phase on the ML deployed model.

(*) DEEL Certification Workgroup. 2021. White Paper - Machine Learning in Certified Systems. Toulouse, IRT St Exupery, 2021.

(**) First usable guidance for Level 1 machine learning applications (EASA Concept Paper, 2021)

These activities should be covered by traditional methodologies and tools, however some specific activities may be needed to verify the ML deployed model:

ML model stability - Analysis of the possible small perturbations in both design/operational phase to support the demonstration that the ML model is robust (will not activate unintended behaviour) to adverse inputs or conditions within the ODD, should they be accidental or intentional (need for specific dataset?). With respect to EASA concept paper in its objective IMP-08: *The applicant should provide an analysis on the robustness (or stability) of the inference model* by performing both abnormal range tests and adversarial tests. Adversarial testing consists in defining corner cases (not based on the requirements) that may affect the AI/ML component expected behaviour.

Distribution shift - Dataset shift is a challenging situation where the joint distribution of inputs and outputs differs between the training phase and the operational use (indeed the training dataset distribution may be modified when augmenting the dataset to include corner cases). In this case the generalization to the true distribution may alter the ML model behavior (addressed by the design?).

HW failures detection and mitigation - Analysis of the impact of hardware faults on the ML model performances.

Unintended function detection - Provide justification that the ML implementation process does not adversely impact generalization capabilities or result in any unintended behavior by the ML Item.

A.5. Gap analysis and challenges

In this section, we list the current standard (EUROCAE/ED-12C or RTCA/DO-178C) objectives that are due to qualify a software item and identify the gaps and the related challenges to integrate a ML model development in a classical software item development. This work is software oriented assuming that the ML model is implemented as a software item. The same analysis should be done to adapt the gap analysis to specific hardware development analysis.

A.5.1 Planning (development and certification plans)

DO-178C objectives for planning :

(a) all the transformation steps from the designed item to the deployed item are defined

Gaps - All the activities specific to ML development will have to be described in the development plans (SDP - SW Dev Plan, SVP - SW Verif Plan, SCMP - SW Conf Mgt Plan, SQAP - SW Quality Assurance Plan). The certification plan (a.k.a. PSAC - Plan for Software Aspect of Certification) will have to be completed accordingly.
Challenges - None, except that the assurance objectives specific to the ML development should be added in the planning documentation (PLAC - Plan for Learning aspect of Certification). Structured argumentation patterns (assurance cases) could be proposed to describe the learning assurance activities supposed to cover the new ML standard objectives.

A.5.2 Development

(a) DO-178C objectives for requirement capture:

(a) Implementation requirements are defined

(b) Interfaces are fully defined to support HW integration and SW integration

(c) Derived requirements are defined and justified

(d) Traceability to input requirements is demonstrated (transparency)

(e) Requirement Standard adherence is demonstrated (consistency)

Gaps - Most of the existing methods/tools are reusable to cover this requirement capture phase. There are some specificities due to ML model implementation: the designed ML model should be fully described to such an extent that its semantics is unambiguous and does not leave any room for interpretation during implementation whatever the target. For instance a Neural Network description should contain its complete architecture description (e.g. number of layers, number of neurons per layer, hyper-parameters weights) but also the internal operations used for the inference in the building environment (e.g. convolution operations implemented by the native framework such as Tensor flow or PyTorch). They are of paramount interest when the model inference is to be exactly replicated in the operational environment.

Challenges - The format of the ML model description should be carefully defined to consider all the necessary information for a correct implementation. There is no such format available at this point of time. As of today, the current frameworks e.g. TensorFlow, PyTorch, CAFFE) exports a format that does not express the entire semantics of the ML model (e.g ProtoBuf, ONNX) and that utilizes their own layer of services (e.g. padding, average pooling). Therefore it is not possible to guarantee that the model inference in its operational and native environment will be strictly identical. These model semantic discrepancies can be detrimental to the expected performance of the ML model and source of unintended behavior of the item it belongs to. The challenge will be to find a format that is producible from all the native platforms and that holistically describes the semantics of the ML model. What is missing today in the ML Model Description output from the design phase is: readable/auditable data format, network structure completeness, only inference data is required, readable Hyper Parameters, layer's services specification.

(b) DO-178C objectives for implementation design:

   (a) Architecture and LLRs are defined
   (b) Design Standard adherence is demonstrated (consistency)
   (c) Determinism is acceptable
      i. Memory: implemented item is compatible with memory resources
      ii. Timing: implemented item is compatible with host processor resources

Gaps - These traditional activities are also applicable to the implementation of the ML model. The gaps are in the ML model description and the optimisation activities that are specific to ML development processes.

Challenges - Though some methods and tools regarding the conversion and optimization of NN are yet available as part of COTS framework libraries (e.g TensorFlow Lite), their use in the context of embeddability is still part of the research field (refer to coming ERTS2022 paper from the ANITI Certification Mission sprint on Optimization). The challenges are related to the use of optimisation techniques like network pruning, parameter quantization, algebraic optimization, batch-normalisation and convolution fusing or knowledge distillation are of interest. However the way they alter the ML model behavior and performances when used in the implementation phase (meaning with no retraining) is still not mastered.

(c) DO-178C objectives for code and integration

   (a) Code is developed from the implementation requirements and the designed architecture
   (b) Code is verified against the requirements (correctness)
   (c) Traceability to implementation requirements is demonstrated (completeness)
   (d) Coding Standard adherence is demonstrated (consistency)
   (e) Worst cases are mitigated

- Memory: the worst case is computable and mitigation is designed
- Timing: the worst case is computable and mitigation is designed

(f) Executable code is developed from the source code with identified means

Gaps - The same objectives remains applicable however methods/tools are highly dependent on the targeted platforms. If traditional means can be used when installation on a classical CPU target, they have not been analysed at this stage for new targets (e.g. TPU, GPU, many cores platforms).
Challenges

- Librairies: the use of the target librairies as part of the embeddable product (e.g. CUDA libraries for a nVDIA GPU) could be really challenging if they are not enough documented. Indeed the applicant will have to demonstrate that their development process satisfies the applicable standard objectives.
- Semantics preservation: the demonstration that ML model semantics is preserved may be needed in the ultimate phase of transformation between source and object code for safety-critical applications.

A.5.3 Verification

DO-178C objectives for verification

1. Executable code is verified to detect inconsistencies with the implementation requirements and the interfaces
2. Executable code is verified to be robust with its requirements and interfaces
3. Requirement-based verification means are developed
   (a) Functional coverage is demonstrated
   (b) Structural coverage is demonstrated
   (c) Data/Control coupling is demonstrated
   (d) Verification reproducibility is demonstrated
4. The verification environment is representative of the operational environment
5. Unintended behavior
   (a) Unused function are identified and safety impact mitigated
   (b) Any additional code inserted by the transformation steps is identified safety impact mitigated
   (c) Derived requirements functional and safety innocuity is demonstrated
   (d) Injection of errors are prevented (standard use are verified)
   (e) Numerical representation error

Gaps - All the objectives are applicable. The relevance of the structural coverage and the data/control coupling can be challenged in the context of the ML development.
Challenges

- Test coverage demonstration: Classical empirical methods (analyses, review and tests) are still usable to verify an item based on its requirements. Massive testing can be used, based on statistical considerations with respect to the probabilistic aspect of the true distribution of the input space and the ODD definition. The main challenges (completeness and representativeness of the test coverage) will increase with the problem dimensions.

- Formal verification scalability: The ML techniques, due to their intrinsic mathematical nature, give the opportunity to emphasize the use of formal methods to formally verify the model properties. However at this point of time, these methods are limited to models resolving low dimensional problems.

- Test representativeness: It shall be noted that the verification demonstration is only valid in the operational context of execution. If any verification activities are performed in a different context (for instance during the native design phase), it should be proved that the verification environment is representative from the operational one in order to take credit of the results for the certification argumentation.

- Determinism verification: The memory and timing determinism are operational constraints that should be fully specified by the previous phases (at system or design level at the latest). According to the platform selection, it may be challenging to get the worst cases estimation and measurement on new targets.

- COTS use: The use of runtime libraries (e.g. libraries, drivers, OS) may be challenging depending on the level of information/demonstration provided by the supplier.

- Tools qualification: The tool qualification may be challenging depending of the level of information/demonstration provided by the supplier.

The robustness challenges are well defined in the DEEL WP(*) from a system viewpoint. If we reduce the scope to the item implementation phase, we have to consider the following topics:

- Robutness to small perturbations: how to measure similarities between normal and abnormal inputs in high dimension (for instance image)? Metrics for similarity should be adapted for all problem dimensionality.

- Robustness metrics: adversarial attacks/defence (TBD WP DEEL page 83)

- Out of Distribution Detection (OODD) in high dimension

- Robustness guarantee: property preservation by formal methods is not scalable to high dimension problems

- Robustness assessment wrt hyper parameters of the ML model: The model's behaviour should be robust to small perturbations of its parameters (e.g. the weights of a deep network) for example due to numerical truncations required by the target hardware on which the model is embedded. Defining appropriate metrics in parameters space poses serious challenges.
- Corner case detection: robustness can also be qualified with regard to the ODD as the capability to detect where a ML system works correctly and where it fails. Providing tools for searching corner case domains is also a challenge.
- Unintended behavior: traditionally addressed by structural coverage analysis, one can think to what extent this technique is applicable to ML algorithm implementation. Indeed, there are no effective model coverage techniques and agreed-upon metrics at this point of time.
- Tolerance to HW failures: what are the impact of the HW failures on the model behavior?

CONFIGURATION/CHANGE MANAGEMENT/PROBLEM REPORTING

DO-178C objectives for configuration/change management and problem reporting

1. The item and its lifecycle data are uniquely identified
2. The item and its lifecycle data are consistent (baseline)
3. All the data needed for replication of a certified item are configuration controlled
4. Any data changes are tracked and managed
5. Data integrity is always preserved
6. Any detected deficiency is captured, recorded and analysed for impact assessment

Gaps - This section will be completed in the final version of the document.
Challenges - This section will be completed in the final version of the document.

A.5.4 Process assurance

DO-178C objectives for process assurance

1. Independently guarantee that processes are consistently applied and output data are compliant and recorded
2. Any process deviation are detected, tracked, evaluated an accepted

Gaps - This section will be completed in the final version of the document.
Challenges - This section will be completed in the final version of the document.

Chapter B

ML implementation standardized process (ED-XXX/AS6983)

This section refers to the state-of-the-art (SOTA) parts of this document. The section C was introducing the ML implementation process, making a gap analysis with traditional development process and identifying the emerging challenges. This section D was proposing a very first certification argumentation (using assurance cases with GSN) with respect to the first work of the EUROCAE/SAE joint working group WG-114/G-34. The delivery for batch 2 is going further in the description of the ML development process assuming the last release (draft 4) of the ED-XXX/AS6983 produced by WG-114/G-34. As per the W-shaped process flow (See Figure B.1), this section will concentrate on the descending part of the second V, describing:

- the standardized implementation process enabling the deployment of the ML-based subsystem on hardware and/or software items,
- the argumentation to respect the certification constraints provided by the AS6983 development assurance objectives. The argumentation is developed using the assurance case concept (GSN format).

The other part of the second V (covering integration and verification activities) will be addressed in the batch 2.

New standard needed - In the airborne context, there is no possible certification of a safety-critical system as long as it is not demonstrated that this system safely performs its intended function in its operating environment according to any foreseeable conditions. In the classical way of developing, the system design follows an ED-79/ARP4754 process design whereas associated software and hardware item are developed to the ED-12C/DO-178C and ED-80/DO-254 standards. These standards have been recognized as acceptable means of compliance to regulation requirements for a long time. When it comes to the use of Machine Learning (ML)-based systems, there is no such recognized guidance and the new standard (still ongoing draft) from the joint standardization group WG-114/G-34 of EUROCAE/SAE (ED-XX/AS6983) should be used.

ML constituent (MLC) concept - This standard introduces novel learning assurance objectives covering the whole spectrum of the design and development of a system incorporating some ML technique. However we will only focus on the implementation aspect of this ML-based subsystem also known as *ML constituent*. The WG-114 introduced the ML constituent concept to fit the industrial needs requesting a maximum of flexibility to possibly host the ML constituent on a combination of HW and/or SW items. From an airborne standpoint (AS6983 also covers ATM/ANS domain), the ML constituent implements the ML-based function that is integrated into a classical avionic system. It means that it contains the pure ML models along with the necessary pre/post processing aspects that are needed to complete the function design. The ML Constituent is a kind of subsystem (from an ARP4754 view) that needs hardware and/or software items (or a combination of these items) to be implemented.

MLMD definition - The main assumption of this chapter is that the ML model has been designed and its performance has been verified to satisfy the system needs (for the sake of simplicity, we will talk about a ML model providing that the design may require the development of several models to design the function). The output of this phase is a description of the ML Model that will be used for the implementation phase. As we focus on the embeddability constraints, we will only describe the implementation part of the process and proposes techniques that enable the compliance to the standard.

Implementation requirements - The AS6983 draft 4 identifies three main requirements with respect to the implementation process:

1. The semantics of the trained model should be preserved on the final hardware platform. This entails in particular that the trained model must be formally and unambiguously defined (in what we call an adequate format) in order to facilitate the MLC implementation.
2. The implementation process should support several ways to deploy the MLC on several HW or SW items. Indeed, not all designers will deploy a model the same way or will choose the same type of item. This entails in particular that the format should allow several types of deployment such as distribution, parallelization or pipelining.
3. The item development has to follow the usual aeronautical development standards (e.g. ED-12C/DO-178C for software).

We will propose an approach compliant with the three former objectives.

B.1. Standardized implementation process

B.1.1 First level

The first level of the standardized process is extracted from the ED-xx/AS6983 draft 4 that was released early December 2022 (See Figure B.1).

The workflow is a W-shaped process. This document focuses on the second V of the workflow which supports the implementation phase. The descending part of the V deals with the development considerations:

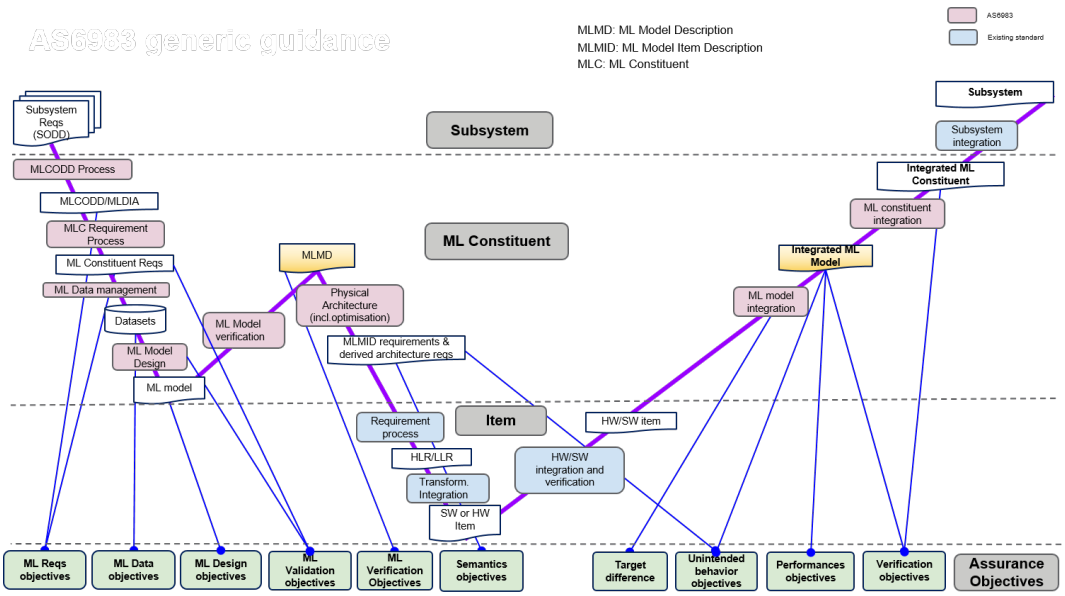

Figure B.1: ED-xx/AS6983 Standardized Process Workflow

1. ML Model Description (MLMD): At this stage the design phase is complete, the ML model is frozen and is not supposed to experience any further training activities. This sequence is obviously theoretical, because in the real life design and implementation are performed in an iterative way, since the correct design is attained. For safety-related component, the implementation process should not alter the safety/functional/operational properties of the ML model obtained by the design process. This means that the preservation of the semantics is ensured and the implemented model is fully verifiable for conformity to regulation requirements. For this purpose, the ML model has to be explicitly and fully described with no possible interpretation. It should permit an exact replication on a software/hardware target (approximation is possible with acceptable criteria). This is the reason why, we have to select a format allowing for a full description of the ML Model semantics. It is the objective of the FA13 to specify and select the adhoc format among those available on the market.

2. Physical architecture: This activity is a major adding from the SOTA delivered in the batch 1. The SOTA was talking about a ML-based item though now we are speaking about a ML constituent. As introduced earlier, the ML technique introduces a kind of ML-based subsystem before the item level and therefore an activity of architecture is needed to decompose the ML constituent into the items used for implementation purposes.

3. Optimisation (optional): With respect to the optimization techniques described in the SOTA, only the operations that are applicable to model description is kept: pruning, quantization, algebraic optimization or fusing batch normalization and convolution.

4. Item development: The development of the items, that are necessary to the ML constituent

implementation, fully relies on the use of the existing guidance from the item development standards (e.g DO-254 or DO-178C).

B.1.2 Second Level - ML Model Description (MLMD)

Refer to FA-13 deliverable for a complete description of the MLMD format. Actually in this section we will focus on considerations that are mandatory to fulfill the certification constraints. The MLMD should include all the necessary information to exactly replicate the designed ML model, meaning that the semantics of the ML Model (graph, parameters representations, operations, dynamics) is fully expressed and reproducible on a specific target. Actually, the AS6983 also allows for an approximate replication of the ML model, in this case the applicant should demonstrate that the difference is acceptable, that is to say not detrimental to any system safety requirements.

The MLMD should contain the following information:

- Architecture elements (explicit description of the graph, parameters and operations)
- Description of optimization operations (e.g. quantization, pruning)
- Execution order of operations
- Additional information (e.g. remarks from the developer, characterisation of the network, replication criteria)

B.1.3 Second Level - ML Model Physical architecture into items (incl Optimisation)

Refer to Figure B.2. The aim of the ML physical architecture process is to give the designer the capability to distribute the model(s) execution onto several items (hardware, software or even a combination thereof). Indeed, for memory or timing optimization reasons, one can think about optimizing the inference or/and splitting the execution on several targets. However these transformations cannot be detrimental to the safety and functional performance that has been required in the specification of the ML function. Therefore the AS6983 draft standard requires some additional activities to ensure that the design model properties are preserved.

- *Inputs* -Actually, the inputs of the ML physical architecture process are the outputs of the design process that produces a ML model that was verified to satisfy the MLC requirements:
  1. ML Model description: See G.1.2.
  2. Performances: the design activity has produced a ML model that was verified to meet the MLC required performances. These performances should be exactly reproduced in case of exact replication otherwise some performance degradation can be acceptable in case of approximate replication. In the latter case, the epsilon should be bounded and not be detrimental to the system safety objectives.

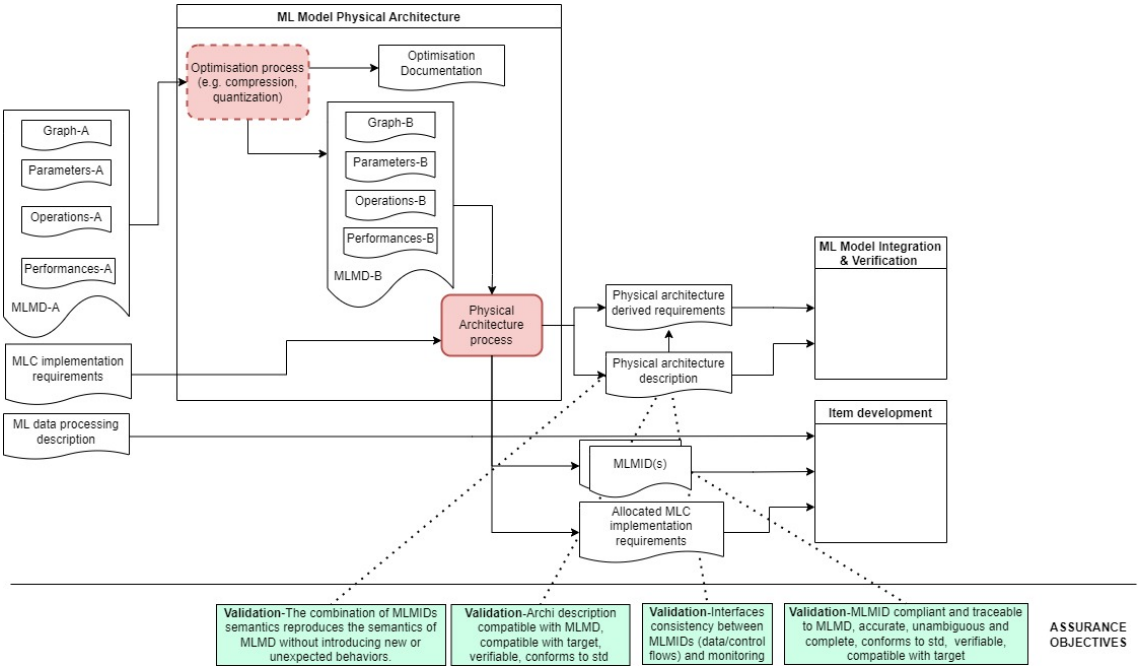

Figure B.2: ED-xx/AS6983- Physical Architecture

3. MLC implementation requirements: The MLC design phase has captured all the implementation constraints flown down from the allocated system requirements. These constraints are linked to operational performances (such as timing or memory footprints that the embedded ML function should respect when integrated in the system) or interfaces to the hardware platforms that has been selected to execute the ML inference.

- *Activities* - The activities are two-fold:

1. Optimisation: this activity is optional. Indeed some particular implementation constraints that have not been covered by the design activity may be addressed here. For instance compression may be necessary to reduce the memory footprint of the ML model or quantization to use the data representation optimised for the hardware platform and leading to significant reduction of the inference timing. In both cases, the semantics of the ML model is changed and can potentially impact the ML model properties.
2. Physical architecture: this activity is key in the implementation because it is to find the best use of the hardware resources (timing and memory capabilities) to satisfy the specified operational performances of the ML-based function without impacting neither functional nor safety requirements. This may require the decomposition of the ML model into several HW or SW items(*) whose execution is performed by identified platform resources. Each item is specified by an allocated part of the

MLMD which is called the MLMID (ML Model Item Description). The interfaces between the items (data and control flows) are defined so that their collaboration makes the intended function and preserves the ML model properties. One can think about the decomposition of the ML model into ML model elements (a ML model element can be either a part of the ML model graph, e.g. a convolution layer, or one ML model among several ML models) with each ML model element specified within a MLMID.

In the example extracted from AS6983 draft 4 section 7 (figure B.3), the MLMD is composed of three model elements. The first and part of the second are allocated to one SW item, whereas the third and other part of the second are allocated to a second SW item. All SW items are hosted on the same target (e.g. NXP T1040).

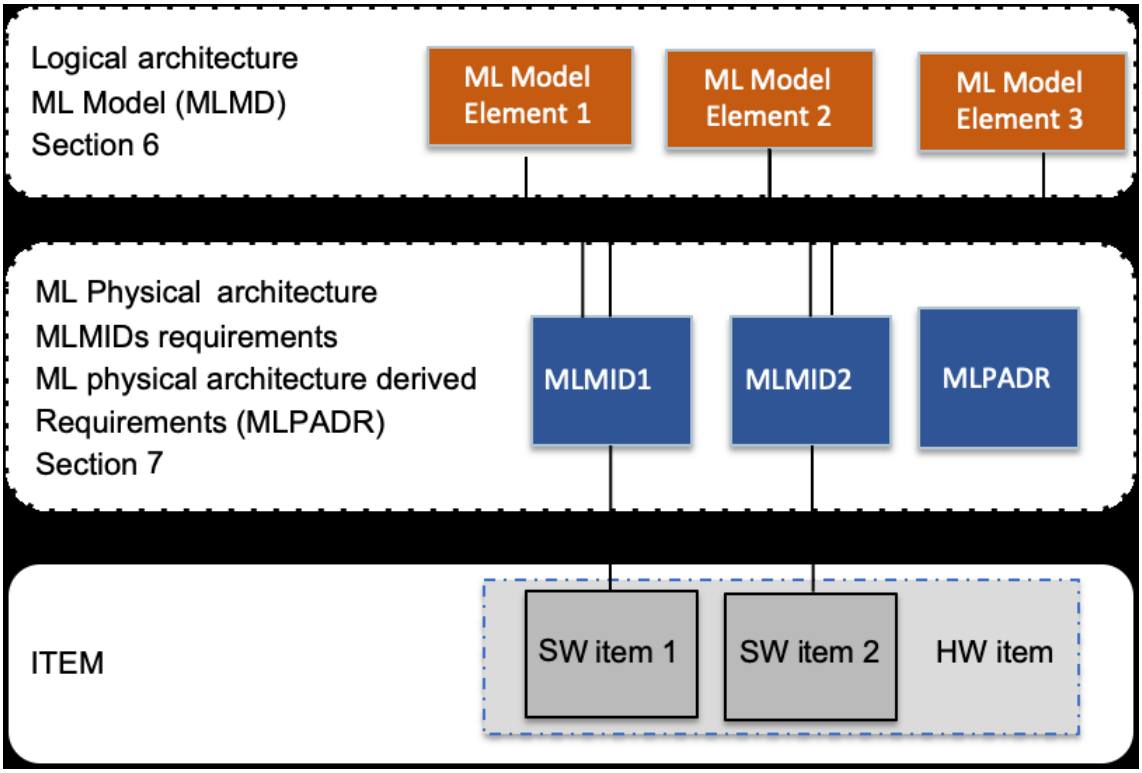

Figure B.3: ED-xx/AS6983- Example of architectural breakdown

In addition, the physical architecture process can produce some emerging behaviour (e.g. local robustness handling, responses to failure conditions, specific protocol implementation between 2 items, device initialization) that should be captured within derived requirements and a rationale to justify their existence. This is the role of the MLPARD (MLPADR stands for ML Physical Architecture Derived Requirements).

(*) item (definition by ARP4754A): A hardware or software element having bounded and well-defined interfaces.

- *Outputs* - The outputs of the physical architecture activities are:

    1. The ML physical architecture description
    2. The ML physical architecture derived requirements
    3. The MLMID(s)

B.1.4 Second Level - Items development

Once all the items that contribute to the ML models inferences are specified (MLMID and ML physical architecture derived requirements) then each item is developed to existing development assurance standard (e.g ED-12C/DO-178C for the SW items and ED-80/DO-254 for the HW items). These standards covers the design, coding, integration and verification of the item with respect to its input requirements. There is no specific activities due to the ML nature of the function, therefore this section will not be further elaborated. Refer to existing standards for details.

B.2. Certification considerations - Conformity to AS6983 objectives

For the batch 2, the demonstration focuses on the development objectives. This section will be further developed in batch 3 to cover the integration and verification processes. This section is based on the assurance case (cf figure B.4) that develops the certification argumentation using the AS6983 guidance (draft 4).

The argumentation develops a strategy that splits the overall process (W-shaped) into 2 main goals:

1. Goal-MLC design (*The MLC specification is a satisfactory refinement of the allocated system requirements for the selected iDAL*): this aspect is not addressed in EC7.

2. Goal-MLC implementation (*The integrated MLC is a satisfactory implementation of the MLC requirements for the selected iDAL*): this aspect is the EC7 scope.

This section focuses on the MLC implementation aspects that are developed into 3 main goals (as per AS6983 guidance). These 3 goals have to be satisfied in order to fulfill the Goal-MLC implementation:

1. Goal-Module(*) MLC Physical Architecture (*The decomposition of the MLC architecture into item(s) (MLMIDs) is a satisfactory refinement of the MLMD and the implementation requirements for the selected iDAL*): This goal is elaborated in figure B.5 and detailed in the next paragraph.

2. Goal-MLC items development (*Each item is developed according tho the MLMID and any derived requirements for the selected iDAL*): The development assurance standards ED-12C/DO-178C and ED-80/DO-254 are applicable and cover the whole item development.

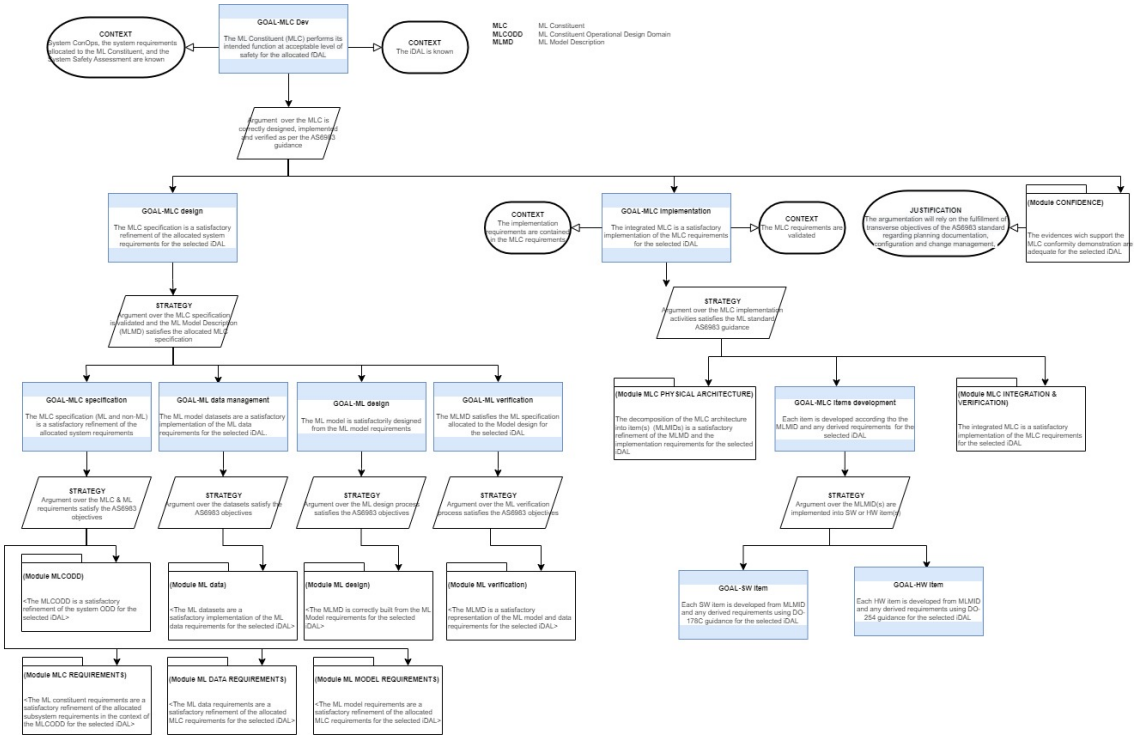

Figure B.4: ED-xx/AAS6983 Assurance Cases - MLC upper level

3. Goal-Module MLC Integration and Verification (*The integrated MLC is a satisfactory implementation of the MLC requirements for the selected iDAL*): This part of the assurance case will be detailed in the Batch 3.

(*) Module means that the goal is further decomposed

B.2.1 Certification constraints on the MLC physical architecture

This paragraph refers to figure B.5. The goal *MLC Physical Architecture* is satisfied when both goals *architecture description* and *architecture validation* are fulfilled.

The strategy to demonstrate the first goal *architecture description* applies on all the architecture elements which should meet the following objectives:

- GOAL-Architecture description (*The ML physical architecture is developed from the MLMD*): The MLMD is decomposed into items to deploy the ML model inference on the target resources.

- GOAL-MLMID (*Each MLMID is developped from the MLMD and physical architecture*): There is one MLMID by item. The interface between MLMIDs are described through data/control flows.

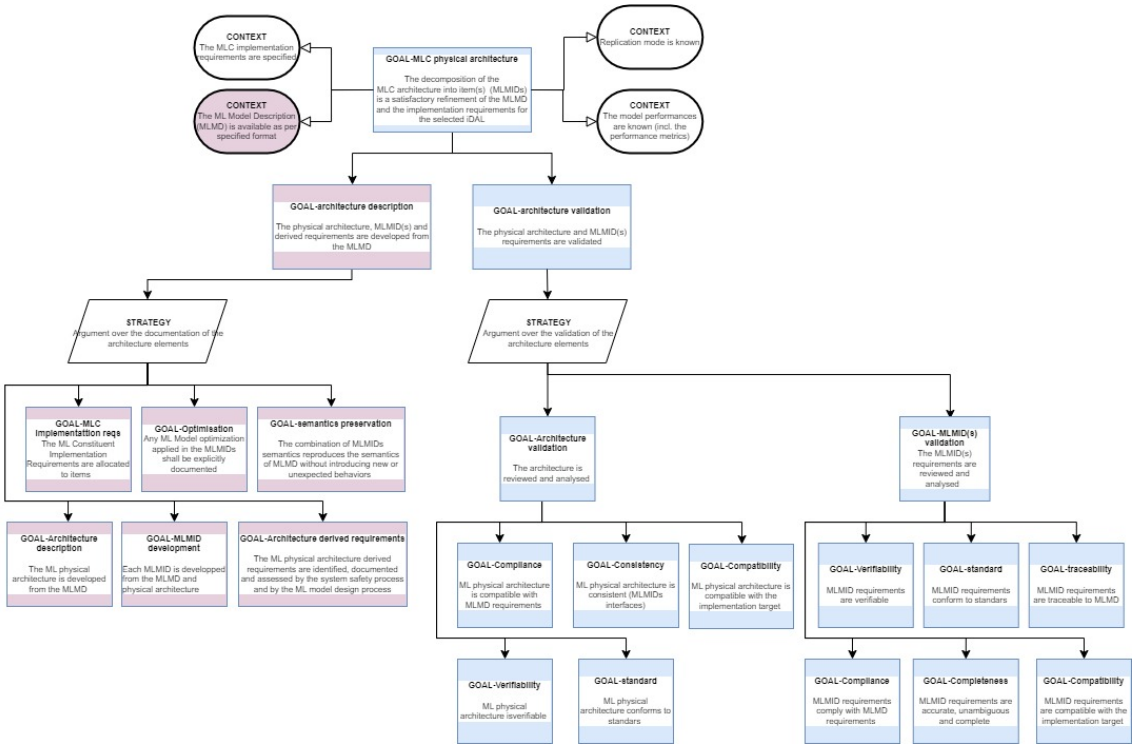

Figure B.5: ED-xx/AAS6983 Assurance Cases - MLC upper level

- GOAL-Architecture derived requirements (*The ML physical architecture derived requirements are identified, documented and assessed by the system safety process and by the ML model design process*): any emerging function/behaviour in between the items are specified as derived requirements.

- Goal-MLC Implementation requirements (*The ML Constituent Implementation Requirements are allocated to items*): There are specific requirements captured at MLC level that were flown down from the system requirement and that are specific to the HW resources. They are allocated to the items.

- GOAL-Optimisation (*Any ML Model optimization applied in the MLMIDs shall be explicitly documented*): Any optimisation that were applied to the ML models are described in the MLMIDs. These transformations are supposed to preserve the ML model semantics.

- GOAL-Semantics preservation (*The combination of MLMIDs semantics reproduces the semantics of MLMD without introducing new or unexpected behaviors*): This is the stringent constraint of the AS6983 objectives. The full preservation of the MLMD semantics should be evidenced (exact replication) or at least the impact of the non-preservation should be mastered and it should be proved that this impact is not detrimental to the safety requirements allocated to the MLC.

The strategy to demonstrate the second goal (*architecture validation*) applies on all the outputs of the architecture activity. The goal is split into 2 sub-goals:

1. GOAL-Architecture validation (*The architecture is reviewed and analysed*)
2. GOAL-MLMID(s) validation (*The MLMID(s) requirements are reviewed and analysed*)

For the first goal, the following objectives should be met:

- GOAL-Compliance (*ML physical architecture is compatible with MLMD requirements*): The objective is to ensure that the ML physical architecture does not introduce any conflict with the MLMD requirements, especially functions that ensure system integrity (e.g. partitioning schemes).
- GOAL-Consistency (*ML physical architecture is consistent (MLMIDs interfaces)*); The objective is to ensure that a correct relationship exists between the MLMIDs identified by the physical architecture. This relationship exists via data flow and control flow.
- GOAL-Compatibility (*ML physical architecture is compatible with the implementation target*): The objective is to ensure that no conflicts exist with the target resources, especially initialization, asynchronous operation, synchronization, and interrupts, between the MLMIDs and the hardware/software features of the target.
- GOAL-Verifiability (*ML physical architecture is verifiable*): The architecture should not introduce any specificity that would be difficult to test.
- GOAL-Conformance to standard (*ML physical architecture conforms to standards*). The homogeneity of the architecture decomposition is ensured by the use of a design standard. The adherence to this standard should be verified.

For the second goal, the following objectives should be met:

- GOAL-Compliance (*MLMID requirements comply with MLMD requirements*): The objective is to ensure that
  - the MLMID requirements satisfy the MLMD requirements.
  - Derived requirements and the justification for their existence are correctly defined
  - the MLMID reflect the MLMD implementation choices (exact or approximated replication)
- GOAL-Completeness (*MLMID requirements are accurate, unambiguous and complete*): The objective is to ensure that each MLMID requirement is accurate, unambiguous and complete, and that the MLMID requirements do not conflict with each other.
- GOAL-Compatibility (*MLMID requirements are compatible with the implementation target*): The objective is to ensure that no conflicts exist between the MLMID requirements and the hardware/software features of the target, especially the use of resources such as bus loading, system response times and input/output hardware.

- GOAL-traceability (*MLMID requirements are traceable to MLMD*): The objective is to ensure that all the MLMD requirements were developed into the MLMID requirements and that the MLMID does not introduce any unintended behaviour.
- GOAL-Verifiability (*MLMID requirements are verifiable*)
- GOAL- Conformance to standard (*MLMID requirements conform to standards*): The homogeneity of the MLMID description is ensured by the use of a design standard. The adherence to this standard should be verified.

Bibliography

[Daedalean, 2021a] Daedalean, E. a. (2021a). Concepts of design assurance for neural networks (CoDANN).

[Daedalean, 2021b] Daedalean, E. a. (2021b). EASA and daedalean, concepts of design assurance for neural networks (CoDANN) II,.

[Delseny et al., 2021] Delseny, H., Gabreau, C., Gauffriau, A., Beaudouin, B., Ponsolle, L., Alecu, L., Bonnin, H., Beltran, B., Duchel, D., Ginestet, J.-B., et al. (2021). White paper machine learning in certified systems. *arXiv preprint arXiv:2103.10529*.

[EASA, 2021] EASA (2021). EASA concept paper: First usable guidance for level 1 machine learning applications.

[Food et al., 2019] Food, Administration, D., et al. (2019). Proposed regulatory framework for modifications to artificial intelligence/machine learning (ai/ml)-based software as a medical device (samd).

[Han, 2007] Han, C.-H. (2007). International electrotechnical commission. *Electric Engineers Magazine*, pages 29–34.

[Hawkins et al., 2021] Hawkins, R., Paterson, C., Picardi, C., Jia, Y., Calinescu, R., and Habli, I. (2021). Guidance on the assurance of machine learning in autonomous systems (amlas). *arXiv preprint arXiv:2102.01564*.

[Jenn et al., 2020] Jenn, E., Albore, A., Mamalet, F., Flandin, G., Gabreau, C., Delseny, H., Gauffriau, A., Bonnin, H., Alecu, L., Pirard, J., et al. (2020). Identifying challenges to the certification of machine learning for safety critical systems. In *European Congress on Embedded Real Time Systems (ERTS 2020)*.

[Jia et al., 2021] Jia, Y., McDermid, J., Lawton, T., and Habli, I. (2021). The role of explainability in assuring safety of machine learning in healthcare. *arXiv preprint arXiv:2109.00520*.

[Koopman et al., 2019] Koopman, P., Ferrell, U., Fratrik, F., and Wagner, M. (2019). A safety standard approach for fully autonomous vehicles. In *International Conference on Computer Safety, Reliability, and Security*, pages 326–332. Springer.

[Matsubara et al., 2021] Matsubara, K., Lieske, H., Kimura, M., Nakamura, A., Koike, M., Morikawa, S., Hotta, Y., Irita, T., Mochizuki, S., Hamasaki, H., and Kamei, T. (2021). A 12-nm autonomous driving processor with 60.4 tops, 13.8 tops/w cnn executed by task-separated asil d control. *IEEE Journal of Solid-State Circuits*, pages 1–1.

[Tambon et al., 2021] Tambon, F., Laberge, G., An, L., Nikanjam, A., Mindom, P. S. N., Pequignot, Y., Khomh, F., Antoniol, G., Merlo, E., and Laviolette, F. (2021). How to certify machine learning based safety-critical systems? a systematic literature review. *arXiv preprint arXiv:2107.12045*.

Confiance ai

Titre:	État de l'art de l'Embarquabilité des composants d'IA de type Machine Learning

Mots Clés: Confiance, contraintes sur les ressources, contraintes temporelles, contraintes de sûreté de fonctionnement, contraints de certification

Abstract: Ce document présente la fiche actione 6 de projet EC7

Title:	Embarcability of Machine learning components state of the art

Keywords: trust, safety, ressources constraints, timings constraints, safety constraints, certification constraints, embarcability

Abstract: Document for the Action Sheet 6 of EC7 projet.

Our partners:

Air Liquide AIRBUS Atos cea Inria
NAVAL GROUP GROUPE RENAULT SAINT EXUPÉRY SAFRAN sopra steria
SystemX THALES Valeo