EC7.21

ACAS-Xu implementation & Certification

?Document reference number for ANR?

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

Document reference: XXX

Contributors

	Name	Organisation	Role
Responsible for the deliverable	Christophe GABREAU	Airbus	Author
Scientific responsible			
Co-authors	Cristian MAXIM	IRT-SystemX	Author
	Eric JENN	IRT-Saint Exupery	Author
	Marie-Charlotte TEULIERES	Airbus	Author

Document Control

Revision	Date	Commentary	Author
v1.0	24/12/23		Christophe Gabreau

Contents

A. Introduction and document structuring

Activity	Document	Software	Authors
21A - Implementation plan	EC7.21 specific activity (Batch3)	N/A	Christophe Gabreau
21B - Design			
21B1 - Quantization	Rapports ACAS XU quantified models quantification (Batch3)	Notebooks and quantified models	Datakalab (Lucas Fischer)
21B2 - Formal verification	ACAS XU quantified models formal verification (Batch 4)	TBD	Augustin Lemesle
21C - MLMD Specification for Acas-Xu Use Case	EC7.18 specific activity (Batch3)	TBD (Batch 4)	Marie-Charlotte Teulieres
21D - Implementation and performance verification	ACAS-Xu implementation description. ACAS-Xu Verification and Validation report (Batch3)	ACAS-Xu implementation	Eric Jenn
21E - Certification argumentation	EC7.17 specific activity (Batch4)	N/A	Christophe Gabreau

B. Implementation plan

B.1. Introduction

B.1.1 Context

In the airborne context, a safety-critical system cannot be certified as long as it is not demonstrated that this system safely performs its intended function under all foreseeable operating and environmental conditions.
This demonstration only holds when the intent, along with the safety objectives, are satisfied in the target environment. When it comes to embed systems based on Machine Learning (ML) technology, the use of formal methods at design level seems very promising to support this demonstration and therefore, avoid massive and costly testing activities on the selected HW platform. In addition, formal verification covers some of the learning assurance objectives from the novel ML standard ED-324/ARP6983 (ongoing work from the joint EUROCAE/SAE working groups WG-114/G-34).
The full paper will extend previous works already reported in (1),(3) and (4). While (1) and (3) were elaborating the design aspect of the development, (4) was focusing on the implementation part by proposing a method to implement a certifiable system with respect to ED-324/ARP6983 objectives. This paper will propose to study the capability to apply this method on several target platforms and to support the exact replication of formally proven ML models. This work has been performed in the frame of the Confiance.ai project[1].

B.1.2 Contributions

The contributions are the methods to develop a capability for an exact implementation. They are supported by the following activities:

- Design the models, i.e. quantify, formally verify the quantified models and provide a formal specification of the quantified models' semantics
- Implement the quantified models, i.e. study the capability of an exact replication by respecting the formal specification of the designed models on several targets
- Contribute to certification, i.e. provide elements of a certification argumentation to demonstrate the conformity with the current recommendations of the WG-114/G-34 joint working group.

The abstract is structured as follows: section II describes the related work in the field of implementation of embedded ML; section III gives a brief description of the use case used to illustrate the approach and introduces the overall implementation strategy; Sections IV, V, and VI respectively address the design, implementation and certification aspects of the development.

[1]Web site: confiance.ai

B.2. Case study and Implementation Strategy

B.2.1 The ACAS-XU case study

The ACAS-XU case study is described in details in (1). Basically, the ACAS-XU system contains a specific function which computes the best advisory maneuver in order to prevent collision between UAVs for a set of input data (geometrical configuration of the ownship and the intruder UAVs). The geometry of the system is given in Figure B.1. The parameters are:

- ρ (ft), the distance from ownship to intruder
- θ (rad), the angle to intruder relative to ownship heading
- ψ (rad), the heading angle of intruder relative to ownship heading direction
- v_{own} (ft/s), the speed of ownship
- v_{int} (ft/s), the speed of intruder
- τ (s), the time until loss of vertical separation.

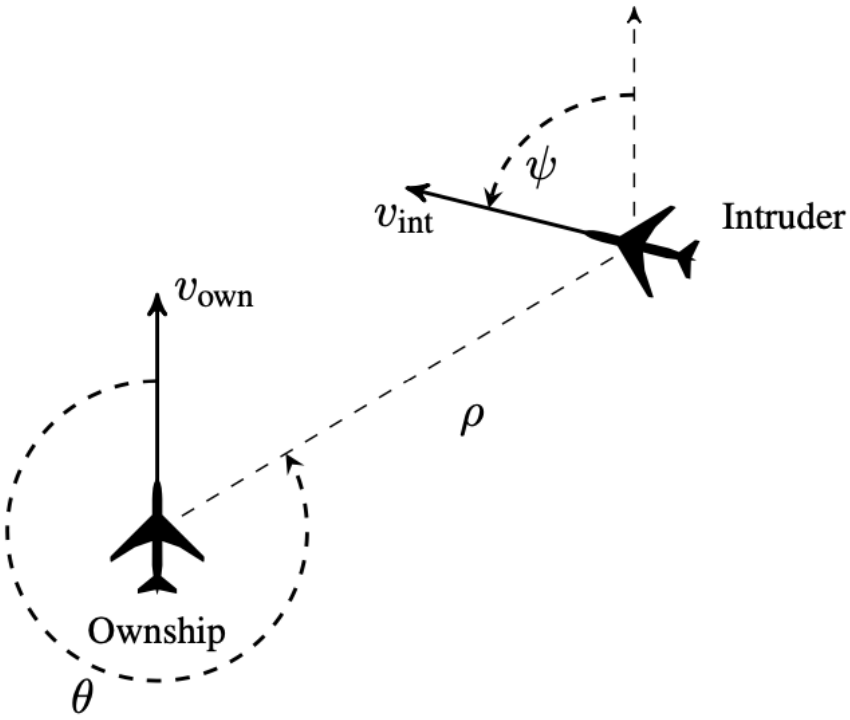

Figure B.1: geometry (5)

The ACAS-XU function is implemented using ML. The ML component, or ML Constituent (MLC) as per WG-114/G-34, is composed of ML model(s) and associated pre/post data processings that can be deployed on one or several *items*. The use of the term "item" complies with the definition given in the existing airborne system development guidance ED-79A/ARP4754A (8).

B.2.2 Implementation strategy

We decided to reuse the safe hybrid architecture that was used to develop the ACAS-Xu ML-based function(1), which is composed of the *NN-based controller* (basically the MLC), the *safety*

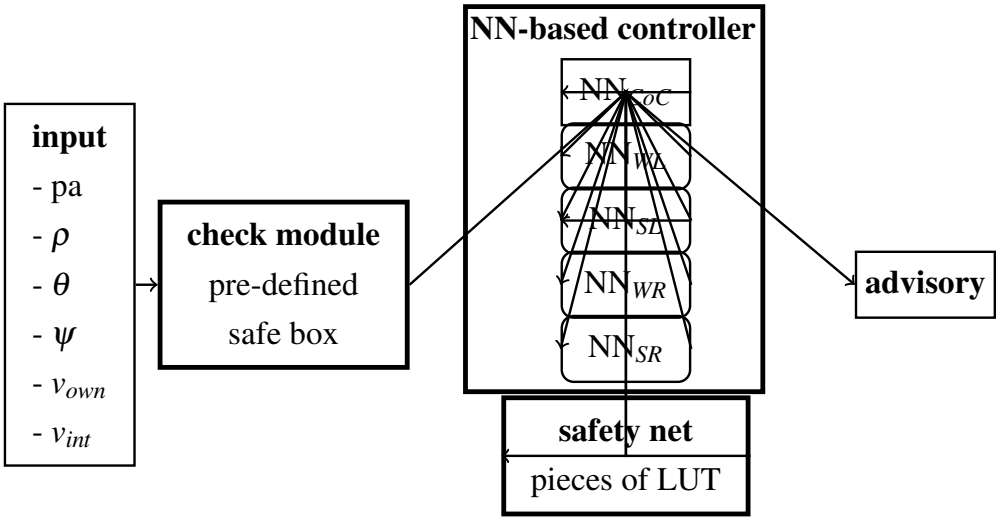

Figure B.2: Hybrid Architecture Overview (1)

net and a *check module* which, in real time, conditions the execution path to provide the correct advisory (cf. Figure B.2). We apply the method proposed in (4) to define an implementation strategy with the following main phases:

- **Specification**: We will specify some requirements (performance, functional, safety, operational) and the associated metrics that can be defined at design level and verified after implementation for performance evaluation purposes. In particular, we will reuse the similarity property defined in (1) (*"decisions of the model are identical to the ones obtained from the tables"*). In addition, requirements have to properly constrain the pre/post processing of the data. This phase will not developed further.
- **Model Design/Quantization**: Among the 45 models designed in the original study (1), we simplify the problem by only considering avoidance maneuvers in the horizontal plan τ=1. The models will be quantified in order to meet the implementation constraints, indeed integer representation is preferred for memory footprint and inference latency reduction purposes.
- **Model Design/ODD partitioning**: The input domain is partitioned into two domains: one domain in which each quantified model is *proved* to hold the similarity property and one domain in which it does not. In the latter case, a safety net is used instead (cf Figure B.2). These proofs are realized by using formal verification methods. At the end, The formally verified quantified models become the reference for the implementation.
- **Model Design/Description**: As per ED-324/ARP6983, the outcomes of the design phase is the ML Model Description (MLMD). This MLMD should express the full semantics of the quantified model to enable the capability of exact replication during implementation. This paper will describe the semantics of the quantified models and stress the gap between the current existing format and what is really needed in the avionic field.
- **Implementation**: This phase starts with the architecture of the ML Model, i.e. its decomposition into multiple ML Model Item Descriptions (MLMIDs), each MLMID corresponding to one item of the physical architecture. Then MLMIDs are transformed into implementation models for the different targets. In this paper, we will consider the *exact* replication of the ML models, i.e. the inference ML model correctly and completely im-

plements the specification of the ML model semantics expressed in the MLMD. Specific techniques (Iree tool with MLIR format) will be used to control the transformations from MLMID to executable code in order to demonstrate the full preservation of the quantified model semantics and support the exact replication.

- **Exact replication study**: According to the obtained replication level (related to the used target), the performance of the implemented models will be verified against the performance of the designed models.
- **Certification argumentation**: We will build an argumentation which demonstrates the objectives defined in the EUROCAE/SAE WG-114/G-34 ongoing standardization work (ED-324/ARP6983). This argumentation will contribute to guarantee that the implementation preserves the model properties and do not introduce any unacceptable unintended behaviours. The argumentation will be developed using the GSN assurance case notation.

B.3. Design

B.3.1 Quantization

Quantization plays a pivotal role in the optimization of machine learning models, with a specific focus on the Data-free method (7), (Edouard Yvinec and Bailly). This method aims to preserve the mathematical function of Deep Neural Networks (DNNs), making it particularly relevant in the context of the ACAS-Xu case study. Here, we consider a subset of five models at τ=1. The quantization process achieves precise activation using diverse calibration methods. It may introduce the usage of bias correction (cf(7) to improve the performance of the quantified models.

We will present the results of quantization along with the reached performance by comparison with the original models.

B.3.2 ODD partitioning

The partitioning of the ODD is realized by verifying where the similarity property holds. This leads to the partitioning of the ODD into 2 classes of input space: where the NN-based controller performs correctly and where it does not (in this case the safety net is used as per Figure B.2). The property is verified using formal methods.
The formal verification activity is performed in the context of the design environment (using specific numerical representation). In order to take credit of this formal verification in the certification argumentation, this verification should either be done on the final implementation model or on any intermediate model *as long as the preservation of the proven property throughout the implementation process of this intermediate model can be demonstrated*. On the ACAS-Xu case study, the models are quantified (all the parameters of the models are converted to 8-bits integer). Using formal methods with int8 instead of FP32 precludes floating point inaccuracy and rounding considerations.
Nevertheless, quantization also brings new layers and operations that need to be correctly handled by the verification tools.

We will present the results of the formal verification on the quantified networks. It will also detail the impact on the verification process and the improvements of the used tools.

B.3.3 Expression of the ML Model Description (MLMD)

As per ED-324/ARP6983 guidance, the MLMD contains the complete description of the model semantics and is the input of the implementation process.
The MLMD is key for the certification because owning the capability to fully describe the semantics of a ML model, should enable the demonstration that the implementation process does not change this semantics and therefore, that the ML model properties which have been verified at design level, still hold. In the scope of ACAS-Xu case study, the MLMD is based on the ONNX format, a format which is not fulfilling the semantics description requirement.

The challenge will be to complete the MLMD with additional information to cover implementation needs (including all operations and approximation fitting hardware constraints, such as quantization).

B.4. Implementation

B.4.1 Physical design

As per ED-324/ARP6983 guidance, this phase aims at defining the appropriate architecture of the MLC supporting the deployment of the ML Models on possibly one or several targets. This implementation activity deals with the both components of the design outcome:

- MLMD part (ML model): The appropriate architecture is designed with respect to the resources available on the selected hardware platform. This activity will identify all the items which are necessary for the deployment of the model(s) onto the target. The MLMD is decomposed into ML Model Item Descriptions (MLMIDs). There is one MLMID per item.
- Non-MLMD part: the specific software parts needed for the pre/post data processing and the selection of the proper model execution (*Check Module* in Figure B.2) according to operational inputs.

We will present the physical design activity that produces the MLC architecture description for each selected target

B.4.2 Items development

This phase aims at implementing each item (coding, compiling, linking, loading and integrating) for the platform resource to which it will be deployed.

B.4.3 Exact replication study

When using complex COTS implementation tool chains, the first option is generally not an option, because the transformation is too complex to be analyzed and demonstrated to be correct, or simply because the details of the implementation are not available. This is to some extent similar to the case of the compilation of source code: except in some very specific cases (e.g, CompCert(6)), the complex series of transformation performed by a compiler cannot be demonstrated to preserve the semantics of the input code.
The solution then consists in verifying the implementation by testing. For testing to be applicable, a correctness criterion must be defined and an oracle providing the expected value is needed. A correctness criterion can be the identity of the final output of the network (the advisory to be

applied), the identity of the last layer of the network (costs provided by the LUT), or the identity of all activations of the network. If we want to ensure a strict preservation of the semantics, the last criterion shall be applied.
Once the criterion is defined, test oracles must be provided. Here, the oracle must predict all activations values. This means that those activation values must be computed, which also means that the execution infrastructure (software and hardware) is also part of the oracle. For instance, the oracle may use TensorFlow running on a x64 PC to compute all activation values.

We will present and compare four different implementations of the ACAS-Xu model: (i) one implementation on an ARM core using the Keras2C tool, (ii) one implementation on the Texas Instruments Jacinto platform, (iii) one implementation on Xilinx MPSoC platform and, finally, (iv) one implementation on the NVIDIA platform. For each deployment platform, focus will be placed on the traceability of the transformations (from the ONNX model to the executable code) and replication test results.

B.4.4 Iree tool capability

With respect with the option 1 defined in the previous paragraph, the efficient execution of neural networks requires the use of complex compilers and execution infrastructures (e.g. TFlite[2], Vulkan[3], CUDA[4]). But to this day, their use typically follows a black-box approach with little traceability between source and executable, and with little control over compilation and resource allocation phases. To address this problem, we have adopted a white-box approach to ML implementation, based on the MLIR/TensorFlow[5] formalism and compiler and the IREE[6] execution environment for embedded systems. Beyond generating efficient implementations, our approach allows:

- Fully tracing the incremental transformation of the source code into efficient implementation binary by exporting the intermediate compiler representation (in textual MLIR/IREE format) after each transformation performed by the compiler. This allows exposing critical transformations such as XLA-level linear algebra optimizations, tiling, buffer allocation, identification of computational kernels (to be executed on GPUs), parallelization, etc., which in turn allows (1) making performance trade-offs by activating/parameterizing compilation passes depending on the application and target architecture and (2) incrementally establishing the equivalence between the source code and the implementation.
- Fine grain profiling using the Tracy tool[7] to gain insight into performance bottlenecks. Its utility is increased by the availability the executed code structured (in MLIR/IREE format).

We will evaluate the output of our compilation chain on embedded multi-core ARM CPUs (CPU-ARMv8.2-A, ARM Cortex-x1, ARM Cortex-A76, ARM Cortex-A55) and consumer GPU targets (Adreno 640, ARM-Mali 78).
Future work includes the use of multiple execution devices.

[2] https://www.tensorflow.org/lite
[3] https://www.vulkan.org/
[4] https://developer.nvidia.com/cuda-zone
[5] https://mlir.llvm.org
[6] https://iree.dev
[7] https://github.com/wolfpld/tracy

B.5. Certification

According to the implementation process constraints (on several targets), different options are possible to show compliance with ED-324/ARP6983 objectives on the implementation part. Either we can demonstrate that the implementation transformations do preserve the full semantics of the quantified models (exact replication) and in this case, credit can be sought for the formal verification performed at design level, or we show that the design performance are maintained, with sufficient verification activities on the target environment to demonstrate that any error introduced by the implementation activities are not detrimental to the safety requirements.

We will develop an assurance case (using the GSN V3 notation as per the methodology developed in (3)) to present the argumentation to fulfill the ED-324/ARP6983 objectives on the implementation part. Will be completed for Batch 4.

C. Design

C.1. Quantization

TBD DATAKALAB (Lucas Fischer)

C.2. Formal verification

Batch 4 delivery (CEA - Augustin Lemesle)

D. MLMD Spefication for Acas-Xu Use Case

Refer to EC7.18 Batch 3 delivery (AIRBUS - Marie-Charlotte Teulières)

E. Implementation and performance verification

TBD IRT St Exupéry (Eric Jenn)

F. Certification argumentation

Batch 4 delivery (AIRBUS - Christophe Gabreau)

Bibliography

[1] Damour, M., de Grancey, F., Gabreau, C., Gauffriau, A., and Ginestet, J.-B. (2021). Towards certification of a reduced footprint acas-xu system: a hybrid ml-based solution. In *Computer Safety, Reliability, and Security 40th International Conference SAFECOMP 2021*, pages pp.34–48, 2021, 978–3–030–83903–1. https://hal.archives-ouvertes.fr/hal-03355299v2).

[Edouard Yvinec and Bailly] Edouard Yvinec, Arnaud Dapogny, M. C. and Bailly, K. Rex: Data-free residual quantization error expansion. arXiv preprint arXiv:2203.14645, 2022.

[3] Gabreau, C., Gauffriau, A., de Grancey, F., Ginestet, J.-B., and Pagetti, C. (2022). Toward the certification of safety-related systems using ml techniques: the acas-xu experience. In *ERTS 2022*, page Session Th.4.C Assurance Certification.

[4] Gabreau, C., Gauffriau, A., Teulières, M.-C., Marandas, D., Pagetti, C., and Maxim), C. (2023). Implementation using ed-xxx/arp6983: the acas xu experience. In *CTIC 2023*.

[5] Katz, G., Barrett, C. W., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135.

[6] Leroy, X., Blazy, S., Kästner, D., Schommer, B., Pister, M., and Ferdinand, C. (2016). Compcert – a formally verified optimizing compiler. In *ERTS 2016: Embedded Real Time Software and Systems*. SEE.

[7] Markus Nagel, Mart van Baalen, e. a. (2019). Data-free quantization through weight equalization and bias correction. In *ICCV, pp. 1325–1334, 2019*, page 1325–1334.

[8] SAE/EUROCAE (2010). Aerospace Recommended Practices ARP4754a/ed-79a- development of civil aircraft and systems.

Title: Title in english

Keywords: Keyword 1, keyword 2

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Our partners: