SONNX: Towards an ONNX Profile for critical systems

Eric Jenn[1], Jean Souyris[2], Henri Belfy[3], Loïc Correnson[4], Mariem Turki[5], Nicolas Valot[6]

[1,5]*IRT Saint-Exupéry, Toulouse, France*
first.last@irt-saintexupery.com

[2]*Airbus Commercial Toulouse, France*
jean.souyris@airbus.com

[3]Thales AVS, *Toulouse, France*
henri.belfy@fr.thalesgroup.com

[4]CEA LIST,Saclay; France
loic.correnson@cea.fr

[6]*Airbus Helicopters, Vitrolles, France*
nicolas.valot@airbus.com

Abstract

This paper presents the objectives and first results of the SONNX Working Group hosted by the ONNX community. SONNX aims at completing the ONNX standard and provide additional artifacts to support the development and certification of critical systems embedding ML algorithms.

1. Introduction

The integration of machine learning (ML) into safety-critical systems presents significant challenges for various reasons including the fundamental and intrinsic probabilistic nature of AI algorithms, the difficulty to explain their results, or more pragmatically, the strong changes to well-established system and software development practices. A large number of publications already discuss these challenges and we invite the reader to refer to [1] for an in-depth discussion about these issues in the context of the development of certificated systems.

In this paper, we focus on one specific activity of the development of a system embedding ML: the one that transforms the representation of a trained model into an executable implementation of the very same model. More specifically, we discuss the case where the model is described using the syntax and semantics defined in the ONNX *de facto* standard "format" (https://onnx.ai/). In this context, the ONNX model is the input specification of the implementation activity. The model may be either interpreted by a tool or translated into some lower level semantically equivalent representation (e.g., some source code) by a tool or a human. In both cases, to be able to interpret the model according to the intent of its designer, its syntax and semantics must be perfectly clear and non-ambiguous. This requirement is applicable to any system, but it is of course critical for systems whose failure may have a critical business or safety impact. In the aeronautical domain, for instance, evidences shall be given to show that the model semantics is actually preserved throughout the implementation phase.

The current ONNX standard, designed and used for "general purpose AI" does not fully satisfy these requirements. Therefore, there is a need to (i) clarify the industrial requirements in that matter, (ii) identify and address the weaknesses of the current standard in a systematic way in order to produce a specific *profile* that would comply with these requirements. We also consider that some of these requirements are domain- and application-specific, so the proposed changes of the standard shall not prevent the use of ONNX in domains where constraints are relaxed. For instance, introducing restrictions on the parameters values for operators of the safety-related profile must not affect the usage of the same operators out of the profile.

In this paper, we describe the objectives, the approach, and the first results obtained by the SONNX working group that is currently developing this "Safety-related ONNX profile"[1].

The paper is organized as follows: Section 2 presents the main objectives and challenges addressed by SONNX, Section 3 gives some first results achieved by the Working Group, Section 4 presents related works, and Section 5 concludes the paper.

2. Objectives and Challenges

2.1. Overview of ONNX

ONNX (Open Neural Network Exchange) is an open-source format designed to represent machine learning models in a framework-agnostic way. It was initiated by Microsoft and Facebook to facilitate interoperability between popular ML tools such as PyTorch, TensorFlow, Keras, and scikit-learn. The ONNX format defines a standard set of operators (OpSet) and a computational graph model that allows models trained in one framework to be exported and run in another. This simplifies

[1] https://github.com/onnx/working-groups/tree/main/safety-related-profile

model deployment and optimization, especially in heterogeneous environments where different hardware targets (CPUs, GPUs, FPGAs, or specialized accelerators) are involved. ONNX models are supported by a wide ecosystem of tools and runtimes, such as ONNX Runtime, Xilinx' VITIS AI, ST Microelectronics STM32Cube.AI, etc.

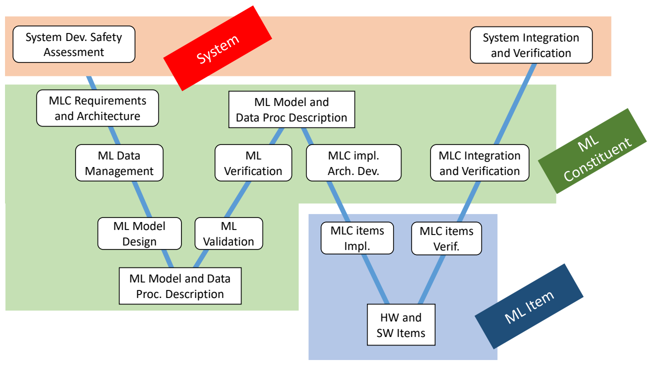

System Dev. Safety Assessment
System
System Integration and Verification
MLC Requirements and Architecture
ML Model and Data Proc Description
ML Data Management
ML Verification
MLC impl. Arch. Dev.
MLC Integration and Verification
ML Constituent
ML Model Design
ML Validation
MLC items Impl.
MLC items Verif.
ML Model and Data Proc. Description
HW and SW Items
ML Item

Figure 1. Overview of the ARP6983 process

2.2. SONNX and the ARP6983/ED-324 MLMD

The SONNX profile is expected to be domain-agnostic. However, as of today, focus is placed on the aeronautical domain and its regulation. This is justified by the very strong representation of industrialists in this domain in the working group, which include representatives of Airbus Commercial, Airbus Helicopter, Thales AVS, etc. This focus is also justified by the relative maturity of the regulatory recommendations standards in this domain. In particular, our work is guided by the EASA Concept Paper [2] and ARP6983/ED-324 [3] currently developed by the joint SAE/EUROCAE WG114 working group.

The SONNX profile tries to address some of the concerns of the ARP6983/ED-324 whose process is recalled on Figure 1.

In this process, the MLMD represent the pivot artifact used to start the development of the ML software and hardware items. It captures hyper parameters, parameters, and the computation graph (composed of a set of interconnected operators). The ONNX computation graph is similar to other data-flow models already used in the aeronautics such as SCADE^{TM}. In both cases, the semantics is the following: a node in the graph is executed once its inputs are available. In the case of SCADE^{TM}, the semantics is synchronous, which is not the case for ONNX.

2.3. The issues and challenges

ONNX is largely used in the AI/ML community, and most of the frameworks and tool produced or consumes ONNX models. This popularity due to several technical and non-technical reasons among which being backed by Microsoft, Meta, Intel AMD, IBM and other key players certainly plays a significant role. The quality of the documentation and software artifacts is sufficient for most usages, but fails short when considering usage in the safety critical or safety related applications. In particular, the specification of some operators given in the documentation is not sufficient to develop and verify an implementation. For instance, the documentation of the convolution operator simply states that "The convolution operator consumes an input tensor and a filter, and computes the output." [2]. In practice, this is usually not a problem since most developer use ONNX as an exchange format between tools but actually rely on some well-documented API to build their models. When it comes to the convolution operator, for instance, PyTorch, gives an extensive description of its convolution operator, including its mathematical description, the formulae to compute the output tensor shapes, etc.

Besides the lack of documentation, or "specification", other issues concern some features introduced in the standard to simplify the developer's activity but raising certification concerns. The shape inference capability of ONNX[3] strongly simplifies the work of the model developer, but it also introduces implicit model modifications that make model verification and traceability between the model and its implementation harder. In the same way, default operator attribute values are also a feature that we would like to avoid in critical systems.
Some elements of the model are explicitly left to the interpretation of the implementer. For instance, concerning operator overloading[3], the documentation states that "*The (domain, name, overload) tuple must be unique across the function prototypes in this list. In case of any conflicts the behavior […] is defined by the runtimes.*". Finally, the order in which operators are executed is also left to the implementer as long as it complies with a topological order of the operators. This obviously makes sense since the semantics of the model does not depend on the actual order, but it may be cases where the model designer wants a specific order to be followed.

The analysis of ONNX "issues" will be extended in the final version of the paper.

2.4. Scope and Requirements

The SONNX standard currently covers only a subset of the ONNX operator set. Choice has been done by the industrial members of the work group in order to support their use cases: real-time object detection using YOLOV8 and MobileNetSSDv2, video tracking using VideoSwin, etc. The complete list is available in the working group's repository[4].

The SONNX profile is build according to 4 main principles:

- **Compatibility with ONNX**, but with stricter constraints

[2] https://onnx.ai/onnx/operators/onnx__Conv.html

[3] https://onnx.ai/onnx/repo-docs/ShapeInference.html

[4] https://github.com/ericjenn/working-groups/blob/ericjenn-srpwg-wg1/safety-related-profile/deliverables/scope/scope.md

- **Clarity and readability** in documentation
- **Determinism in behavior and resource usage**.
- **No ambiguity in specs or graph and operator semantics**.

3. THE SONNX PROFILE

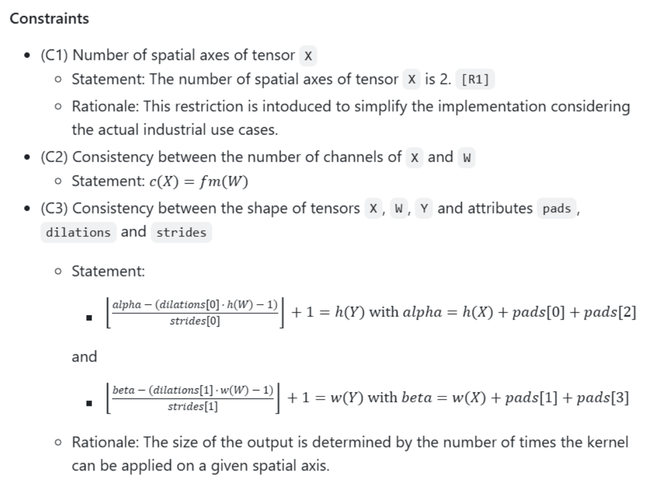
The SONNX profile is composed of three main elements: the operators, the graph, and the file serialization format. In this paper, the question of the file format is not considered.

3.1. The Operators

For each operator, the SONNX profile provides three components: a *documentation* (or "informal specification"), a *formal specification*, and a *reference implementation*.

Documentation. This component describes the behavior of the operator as completely and "intuitively" as possible, with mathematical expressions and illustrations when deemed necessary. As in the original ONNX documentation, a specification of the inputs, outputs and attributes is given, but emphasis is placed on (i) restrictions introduced by the profile and (ii) constraints to be verified for the operator to make sense. An excerpt of this documentation for the `conv` operator is given below. It shows some constraints relating some of the operator's attributes.

Details about the conv operator will be given in the final version of the paper[5]*, along with other examples (e.g., lstm) to show to illustrate the case where operators can be described on the basis of simpler operators.*



Constraints

- (C1) Number of spatial axes of tensor `X`
  - Statement: The number of spatial axes of tensor `X` is 2. `[R1]`
  - Rationale: This restriction is intoduced to simplify the implementation considering the actual industrial use cases.
- (C2) Consistency between the number of channels of `X` and `W`
  - Statement: $c(X) = fm(W)$
- (C3) Consistency between the shape of tensors `X`, `W`, `Y` and attributes `pads`, `dilations` and `strides`
  - Statement:
    - $\left\lfloor \frac{alpha - (dilations[0] \cdot h(W) - 1)}{strides[0]} \right\rfloor + 1 = h(Y)$ with $alpha = h(X) + pads[0] + pads[2]$

    and
    - $\left\lfloor \frac{beta - (dilations[1] \cdot w(W) - 1)}{strides[1]} \right\rfloor + 1 = w(Y)$ with $beta = w(X) + pads[1] + pads[3]$
  - Rationale: The size of the output is determined by the number of times the kernel can be applied on a given spatial axis.

Figure 2. Excerpt of the conv operator documentation

Formal specification. The formal specification is expressed using WhyML, the formal language of the Why3[6] deductive program verification platform. It allows users to write programs or formal specifications, annotate them with properties (such as invariants or pre/post-conditions), and automatically prove those properties using theorem provers. The Why3 environment consists of: the WhyML language for expressing programs and logic, a verification condition generator (VCG) that transforms annotated code into logical proof obligations, a prover interface that connects to automated and interactive provers (e.g., Z3, CVC5, Coq, Isabelle).

WhyML is a strongly-typed functional language extended with imperative features that supports algebraic data types, recursive functions, pattern matching, mutable variables, etc. The Why3 specification serves several purposes. First, it provides a rigorous and mathematically sound specification of the operators that complements the documentation. It is complementary in the sense that it specifies completely the operator whereas the documentation remains informal and usually incomplete in order to remains easily readable and understandable. Second, the formal specification is executable. Therefore, it can be used as an oracle against which a developer can compare his/her own implementation of some operator. This process is illustrated in **Erreur ! Source du renvoi introuvable.**.

Reference implementation. For each operator, we provide a simple and traceable reference C implementation generated from the formal specification. This C implementation is not aimed at being efficient since its main tole is to serve as an oracle for tests. Nevertheless, it may be used in an embedded system, possibly after some rewriting in order to comply with the applicable coding standards.

Examples of WhyML specifications and implementations generated from these specifications will be provided in the final version of the paper.

3.1.1. Numerical accuracy

Numerical accuracy raised numerous discussions in the working group, the main question being: "shall SONNX specify the numerical accuracy of operators?". As of today, the answer is no. Parallel can be drawn between ONNX and SCADE operators but the usage context is very different. In particular, the volume of operations is much larger in neural networks, making very difficult to assess the impact of a computational errors on function performance (e.g., how does an error in an operator affect the final decision?) or, reciprocally, to determine the necessary accuracy of operator or operation level. Moreover, such requirements are rarely defined a priori even in more conventional cases, except in some very specific scenarios.

Therefore, our approach is very pragmatic. Instead of specifying errors, we describe the error estimation method and whenever

[5] The complete documentation of operator `conv` can be found at https://github.com/ericjenn/working-groups/blob/ericjenn-srpwg-wg1/safety-related-profile/documents/profile_opset/conv/conv.md

[6] https://www.why3.org/

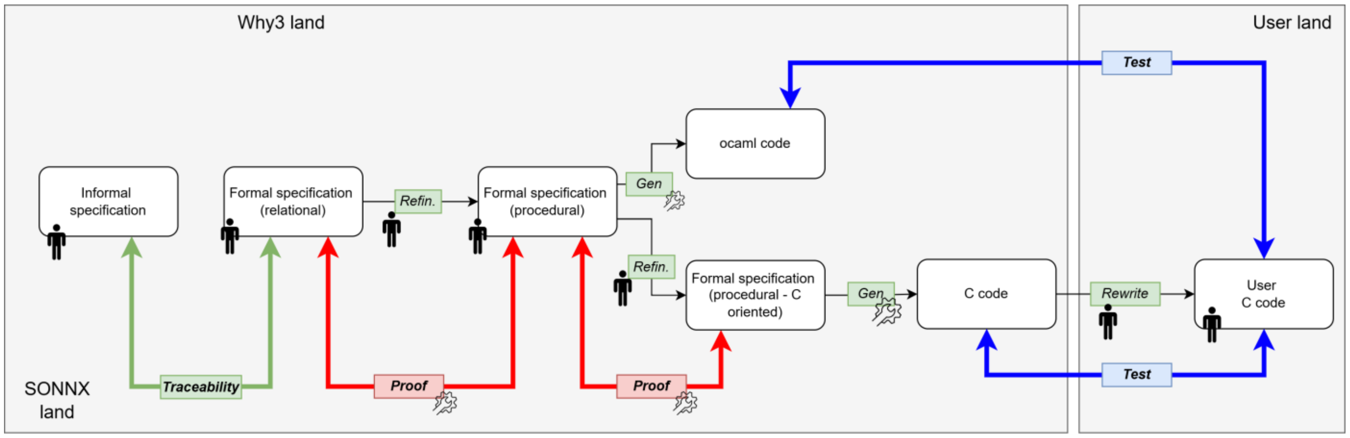


Figure 3. From formal specification to implementation

possible, we aim to provide tools for estimating the error, potentially as an analytical expression or a calculation program (if the error depends on input tensor characteristics, e.g., dimensions and size).

Our approach will be detailed and illustrated on a simple operator in the final version of the paper.

3.2. The graph model

This section concerns the semantics of an ONNX computation graph. It will be developed in the final version of the paper.

3.3. The reference implementation

An example of reference implementation generated from the formal specification will be given in the final version of the paper.

4. Related work

To the best of our knowledge, no other initiative is aimed defining a "safety" subset of the ONNX standard. Nevertheless, several works are aimed at providing a well-defined set of operators. In this abstract, we will only consider the NNEF initiative [5].

NNEF (Neural Network Exchange Format) is an open standard developed by the Khronos Group to enable efficient cross-platform deployment of neural network models. As ONNX, providing a well-defined, extensible model serialization format ensuring interoperability between training frameworks and inference engines. NNEF is particularly geared toward embedded and edge devices, emphasizing portability and hardware acceleration. NNEF comes with a very good documentation. Its serialization format is human readable whereas ONNX relies on Google's protobuf. However, ONNX was embraced early by major ML frameworks like PyTorch and TensorFlow, making it easy to export/import models. The need to interoperability being already satisfied by ONNX, NNEF never got strong backing from those major frameworks and adopted by end-users. Converters exist, but integration is not seamless or well-maintained. NNEF never got strong backing from those major frameworks. Converters exist, but integration is not seamless or well-maintained.

5. Conclusion

The SONNX working group aims at defining a model exchange format compatible with the coming regulations on the basis of the *de facto* ONNX standard. The results will be a refined informal and formal specification of operators and graph, and a reference implementation.

The conclusion will be extended for the final version of the paper.

Acknowledgment

The work described in this paper is the results of the collaborative of the SONNX working group. We thank the ONNX Community for hosting the SONNX working group.

References

[1] F. Mamalet *et al.*, 'White Paper Machine Learning in Certified Systems', IRT Saint Exupéry, Mar. 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03176080

[2] European Aeronautical Safety Agency, EASA Artificial Intelligence (AI) Concept Paper Issue 2: Guidance for Level 1&2 machine learning applications

[3] SAE/EUROCAE, "Process Standard for Development and Certification/Approval of Aeronautical Safety-related Products implementing AI"). ARP6983/ED-324. *To be published*

[4] The Khronos NNEF Working Group, Neural Network Exchange Format, Version 1.0.5, Revision 1, 2022-02-16, https://registry.khronos.org/NNEF/

[5] 'Neural Network Exchange Format', Neural Network Exchange Format (NNEF). [Online]. Available: https://www.khronos.org/nnef/