Introduction

This document contains various elements concerning computation errors and their "impact" on the description of ML models (MLMDs). Currently, those elements are only very loosely organized... Don't hesitate to add your own element and/or comments.

The issue

Floating point computations

- *Floating-point* numbers are encoded on a finite number of bits. Therefore, all real numbers can't be represented exactly. Non-representable numbers must "mapped" (rounded) to representable numbers according to a rounding strategy.
- Concerning operations, the principle (IEEE 754) is that the result of an operation on rounded values shall be the same as the result that would be obtained by rounding the result computed using exact values. The exact statement, from [IEEE754] is the following:

  > "[...] each of the computational operations specified by this standard that returns a numeric result shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that intermediate result, if necessary, to fit in the destination's format"

- In addition, due to the finiteness of the number of bits, overflow and underflow conditions may occur. Conditions may apply on input values to prevent runtime errors. At least, runtime errors shall be signaled. In principle, these preconditions should be be part of the specification of the network (otherwise, there may be conditions in which the model doesn't produce any sensible result).

Non-reproductibility

- Depending on some internal conditions, the same piece of code given the same inputs may lead to different sequences of operations, then different numerical results due to the non associativity of floating point operations. Those internal conditions may be difficult to determine in the absence of a full description of the execution platform. See for instance [Dem-15, Col-15]. In [Dem-15], this effect is studied for a simple parallel summation.
- Some operators ay use random number generation (which sequence depends on a seed).

IEEE-754 and non IEEE-754 data types

IEEE 754 provides a set of guarantees about floating point computations [IEEE-754], however :

- Are all hardware (GPUs, accelerators) implementing the IEEE-754 standard?
  - No: NVIDIA uses TF32.
- Are all hardware devices IEEE-754 compliant (GPUs, FPGA and ASIC accelerators)?
- Some of the data types used in machine learning to not belong to the IEEE 754 standards. For example: BF16 (bfloat16), TF32 (TensorFlow 32 bits)

    - What are the properties of those representations (do they respect the same principle as IEEE 754 numbers?)
    - Should we only allow IEEE 754 standard?

Numerical errors in GPUs

- Precision and performance: Floating point and IEEE 754 Compliance for NVIDIA GPUs -NVIDIA's presentation at GTC 2019 (video, slides)

Numerical errors in CPUs / DSPs

(To be completed)

Numerical errors in ASIC accelerators (NPUs)

(To be completed)

Numerical errors in FPGAs

(To be completed)

Relation between accuracy and ML properties...

This section deals with the relation between computation errors due to (e.g.,) rounding, and the actual expected properties of the ML model. The rationale is that it *may be the case* that bitwise reproducibility is "overkill" with respect to the actual performance of the ML model.

About the effects of floating point errors on ML model performance...

- To what extent is the question of computation errors pertinent with respect to the other sources of errors in Machine Learning algorithms (or "how do computation errors compare to other sources of errors")?
- *(to be completed)*

About accuracy and robustness...

- See Th. Beuzeville about floating point errors and robustness attacks. e.g., [Beu-24]
- *(to be completed)*

About accuracy and quantification...

- There has been a lot of work about quantization. How does it relate to our problem?
- *(to be completed)*

Current Practices and State of the Art

Current practices

- How do we manage numerical errors (inc. floating point errors) in current, non-AI systems?
  - How are requirements about computation errors expressed for those systems?
  - How are those requirements verified?

- Let us consider one operator. The operation is usually specified done using a mathematical formula showing the relation between inputs and outputs (e.g., $y=\pm\sqrt x)$, or $y^2=x$. More often than not, this relations does not depend on the representation of numbers (`float` , `double`...). For instance, a sine function is simply specified at "A function that computes a sine".
  Complex operations are described using a small corpus of basic operators and functions such as the standard four operations ($+$, $-$, $\times$, $\div$), $\surd$, $sin()$, $cos()$, etc. whose semantics is supposed to be well-known.

Standards

All programming languages come with a set of basic mathematical operators, either directly implemented by the hardware or implemented in a mathematical library (e.g., `libm`). Libraries also exist for high-level function such as Basic Linear Algebra (e.g., BLAS, LINPACK,etc.). Some libraries are provided by chip manufacturers in order to take the most out of their hardware (cuBLAS, Intel math lib)

Mathematical libraries

- `libm`
  - The newlib mathematical library does not give the accuracy of operations. (Note that the way mathematical operations are described could be inspiring) In addition, as stated in [Dar-06] "current [as of 2006] libm implementation do not always return the floating point number that is closest to the exact mathematical result. As a consequence, different libm implementation will return different results for the same input, which prevents fill portability for floating-point applications". The same claim can be read in [Gla-24]:

  > The IEEE 754 standard, even in its latest 2019 revision [17], does not require correctly rounded mathematical functions, it only recommends them. In turn, current mathematical libraries do not provide correct rounding, which is the best possible result. Thus, users might get different results with different libraries, or different versions of the same library. This can have dramatic consequences: for example missed collisions in the Large Hadron Collider [5] or reproducibility issues in neuroimaging [13].

- BLAS and other linear algebra libraries
  - BLAS is not accurately rounded. For an accurately rounded BLAS, see e.g., [Cho-16].
  - cuBLAS
    - cuBLAS is designed so that "all [...]] API routines from a given toolkit version, generate the same bit-wise results at every run when executed on GPUs with the same architecture and the same number of SMs. However, bit-wise reproducibility is not guaranteed across toolkit versions because the implementation might differ due to some implementation changes." (cuBLAS documentation, section 2.1.4).
    - The precision of operators is not specified. See e.g, `gemm` in section 2.7.1 .
  - Intel MKL BLAS
    - See e.g.., the `gemm` function: no indications whatsoever about accuracy.

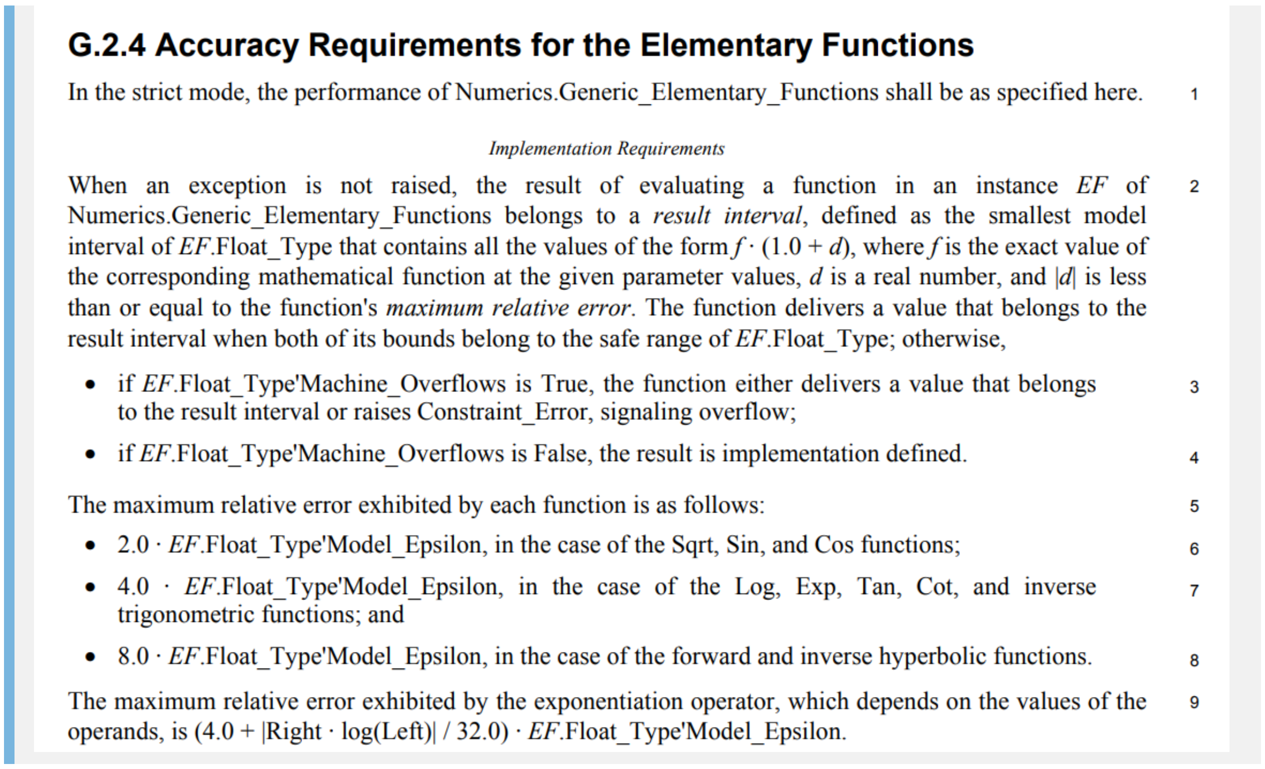
Programming languages

How are floating point operations specified in programming languages? Here are a few examples:

- C
    - In the C99 standard [C99], nothing is said about the accuracy of operator. For instance, for the sin operator, the specification is the following:

```
Description
The sin functions compute the sine of x (measured in radians).
Returns
The sin functions return sin x.
```

    - In C99, "an expression of successive add or multiply operators in interpreted with left-associativity, i.e., `a+b+c+d` is syntactically equivalent to `((a+b)+c)+d)`.[Mul-10]
- Ada
    - The Ada LRM Section G.2.4 gives some requirements about the accuracy for elementary functions":



G.2.4 Accuracy Requirements for the Elementary Functions

In the strict mode, the performance of Numerics.Generic_Elementary_Functions shall be as specified here. 1

Implementation Requirements

When an exception is not raised, the result of evaluating a function in an instance *EF* of Numerics.Generic_Elementary_Functions belongs to a *result interval*, defined as the smallest model interval of *EF*.Float_Type that contains all the values of the form $f \cdot (1.0 + d)$, where f is the exact value of the corresponding mathematical function at the given parameter values, d is a real number, and $|d|$ is less than or equal to the function's *maximum relative error*. The function delivers a value that belongs to the result interval when both of its bounds belong to the safe range of *EF*.Float_Type; otherwise, 2

- if *EF*.Float_Type'Machine_Overflows is True, the function either delivers a value that belongs to the result interval or raises Constraint_Error, signaling overflow; 3
- if *EF*.Float_Type'Machine_Overflows is False, the result is implementation defined. 4

The maximum relative error exhibited by each function is as follows: 5

- 2.0 · *EF*.Float_Type'Model_Epsilon, in the case of the Sqrt, Sin, and Cos functions; 6
- 4.0 · *EF*.Float_Type'Model_Epsilon, in the case of the Log, Exp, Tan, Cot, and inverse trigonometric functions; and 7
- 8.0 · *EF*.Float_Type'Model_Epsilon, in the case of the forward and inverse hyperbolic functions. 8

The maximum relative error exhibited by the exponentiation operator, which depends on the values of the operands, is (4.0 + |Right · log(Left)| / 32.0) · *EF*.Float_Type'Model_Epsilon. 9

Note 1: the result of a computation also depends on the compiler, on the hardware and, also, on the operating system. In the latter case, the same code compiled with the same compiler and ame option may give different results whether it is executed on OpenBSD or Linux since the two OSes initialize the FPU differently (round to double precision for OpenBSD, and round to double-extended precision for Linux). [Mul-10] gives many other useful insights about the way compilers handle floating-point operations. In particular, it lists the pragmas that have an effect on those operations.

Note 2: an interesting quotation from [Mul-10]:

> As pointed out by David Goldberg [...]) all these uncertainties make it impossible, in many cases, to figure out the exact semantics of a floating-point C program just by reading its code. Several examples of strange behaviours are given in the book. One example considers the case where a symmetric function (e.g., a function $f(x,y)$ computing the distance between $x$ and $y$) may be compiled in an asymmetrical way using, e.g., `fma` in such a way that $f(x,y) \neq f(y,x)$, potentially breaking algorithms relying on the symmetry of the function (such as sorting algorithms).

Industrial standards

What do the industrial standards say about computation errors (in space, aeronautics, automotive)?

Space standards

- The LibmCS Mathematical Library for Critical Systems developed for ESA does not provide any data about the accuracy of operations. Special cases are well defined. An example of specification is given below, for the sin operator:

> REQ-BL-0200//GTD-TR-01-BL-0015/T
> The sin and sinf procedures shall evaluate the sine of their argument x in radians.

> REQ-BL-0203//GTD-TR-01-BL-0015/R
> The sin and sinf procedures shall use a minimax polynomial for the calculation.

> REQ-BL-0210//GTD-TR-01-BL-0015, GTD-TR-01-BL-0026/T
> The sin and sinf procedures shall return NaN if the argument is NaN.

> REQ-BL-0220//GTD-TR-01-BL-0015, GTD-TR-01-BL-0026/T The sin and sinf procedures shall return the value of the argument if the argument is ±0.

> REQ-BL-0240//GTD-TR-01-BL-0015, GTD-TR-01-BL-0026/T
> The sin and sinf procedures shall return NaN if x is ±Inf.

- The "ground truth" could be computed using a multi-precision library. For instance, `mpfr` is a library for multiple-precision floating-point computation "which is both efficient and has a well-defined semantics" [Hou-07]

Aeronautical standards

- The DO-178C [DO-178C] doesn't say much besides that the source code must be correct with respect to floating point arithmetic (§6.3.4.f) (the term "floating point" appears only once).
- (May refer to the MOPS, but it is unlikely that we will find anything concerning the allocation of errors to computations.)

Automotive standards

- The ISO 26262-6 [ISO26262] ("Product development at the software level") doesn't say much about floating point computations.

Needs

What are our needs concerning computation accuracy, and **for what purpose**?

Industrial needs and requirement on SONNX

- [**Needs**] The executions of a SONNX model shall be reproducible to support debugging activities.
    - Reproducibility means that given the same inputs, all computations shall give the same outputs.
      The level of reproducibility be defined by (i) a bound on the errors on the outputs, (ii) the supported variability on the target SW or HW.
      For instance, one may require that "the model shall give bitwise identical results when executed on the same HW architecture" or "the model shall give results $\pm \epsilon$ when executed on the same HW architecture", etc.
      Note that this requirement is only about reproducibility, i.e., the results may be perfectly reproducible but completely wrong with respect to the model "semantics".
    - [**Req**] For a given model M, for a given execution platform P, for any valid input x, the maximal difference between multiple executions of the model on x must be bounded by ϵ.
      The bound may depend on the application. It may go from strict -- or "bitwise" similarity -- to similarity expressed by an upper bound on the difference between the activations produced by the different executions.\
- [**Needs**] The implemented model must output the same results as the model used during training in order to gain credit on the verification activities (*to be clarified*).
- [**Needs**] The SONNX model must capture all the elements necessary to implement a model providing ...
- [**Reqs**] If a property is verified on a given source model (e.g, a PyTorch model) M_{org}, the SONNX model generated from the PyTorch model M_{saved} must preserve the property in the sense that if P hold on M_{org}, then P also holds on M_{saved}.
  In particular, this means that the (meta-)model (i.e., the SONNX standard) must preserve the data ensuring the property.
  For instance, using floating point can lead to unsound verification results in the sense that a model that is robust in $\mathbb{R}$ (i.e, verified formally to be robust considering values in R) may not be robust when implemented using finite precision numbers [???]. In this example, the MLMD would have to be implemented in R (if it were possible) for the robustness property to be preserved.

Note (a): "Reproduction" is different from "replication" because the former concerns the relation between different executions of the same model implementation, whereas the latter concerns the relation between the model and its implementation(s).
Note (b): Instead of specifying requirements about accuracy and precision, couldn't we just specify implementation requirements, i.e, the way computations must be done.

Certification "needs"

What are the certification recommendations/objectives that may be impacted by computation errors (determinism, predictability, reproducibility...)?

EASA concept paper

To be completed.

ARP6983

Replication criteria

The concept of "replication criteria" is introduced in [ARP6983]. Two "levels" of replication are defined (the following definitions are taken from the ARP6983 preliminary version):

- **Approximated replication**: ML inference model implemented in software or Airborne Electronic Hardware (AEH) in which some differences with regards to the ML model semantics are acceptable.
- **Exact replication**: ML inference Model implemented in software or Airborne Electronic Hardware (AEH) in which no difference is introduced with regards to the ML Model semantics.

More precisely, in [ARP6983, §6.4.3.6]:

> e. The replication criterion (either exact or approximated) is defined from the ML Constituent requirements and if applicable from the ML Model requirements:
>
> - Exact replication: In this first case, the ML Model description should contain sufficient details on the ML Model semantic to fully preserve this semantic in the implemented ML Model. For example, an exact replication criterion may be the direct and faithful implementation of the ML Model description so that the implemented ML Model meets the same performance, generalization, stability, and robustness requirements.
> - Approximated replication: In this second case, the ML Model description should contain sufficient details on the ML Model semantics to approximate this semantic in the implemented ML Model with a specified tolerance. For example, an approximation metric may be expressed for a given dataset by the maximal gap between the trained ML Model outputs and the implemented ML Model outputs. The corresponding approximation replication requirement may be that this maximal gap should not exceed a given value epsilon.

Those definitions should probably be clarified a bit... For instance:

- The use of the term "replication" is misleading. To "replicate" means to reproduce. But we do not want to "reproduce" the model itself, but to implement it, i.e., to reproduce its behaviour.

- The definition of "semantic" in this context is not clear. The usual meaning of the word "semantic" is "the meaning of something". But what is the "meaning" of a ML model? Furthermore, in the definition, the term semantic is associated with properties such as performance (ML performance?), generalization, stability,... Do these properties relate to the "semantic"?

- "[...] the ML Model description should contain sufficient details on the ML Model semantic to fully preserve this semantic in the implemented ML Model." :

    - (wording) The model cannot "preserve [the] semantic of the implemented model": it is up to the developer to preserve the semantic of the model (which must have a clear semantic).
    - What does "fully preserve" mean, precisely? A naive (but clear) interpretation would be that the semantic of the model is "fully preserved" by an implementation iff, for any input, the outputs of the implementation are **strictly** identical to the ones that would be produced by a strict interpretation of the model. By "strict interpretation", I mean an interpretation strictly compliant with the mathematical definition of the model. Interpretation could be intellectual or based on some mathematical tooling. But since this is usually not applicable in practice, a more operational definition could refer to some well defined, or possibly "reference", implementation. In that case, an implementation would be considered "semantically identical" to the model if, for

the same inputs, it provides exactly the same outputs as the reference implementation. By "exactly", I mean "bitwise identical".

- According to the ARP's definitions, the replication criteria could be expressed in terms of "high-level" (or "end-user') properties such as: ML performance, stability, robustness, etc., which are all properties of the model, not properties of the implementation (as would be the "accuracy", for instance).
- In this context, providing a bitwise accurate implementation would be a means to satisfy *all* replication criteria (except temporal ones).

Numerical accuracy

Numerical accuracy of the model

- Do we have to specify the accuracy of the implementation of the model with respect to the model itself. For example:
  - "The model implementation is correct if all of its activations are at $\epsilon$ to the values that would be produced by a perfect interpretation of the model".
- Would the accurate specified for each element of the model (e.g., all activations) or for a subset of them (e.g., the outputs of the network)?
  For example:
  - Case #1: "The model implementation is correct if all of its activations are at $\epsilon$ from to the values that would be produced by a perfect interpretation of the model".
  - Case #2: "The model implementation is correct if the outputs of the last layer are at $\epsilon$ from the values that would be produced by a perfect interpretation of the model".
- In practice, and besides the trivial case where $\epsilon=0$, will the user be able to specify values for $\epsilon$?

Numerical accuracy the operators

- Accuracy of operators need to be specified, otherwise it will be impossible to assess any implementation.

Nota (a): This is not mandatory or even not possible... As demonstrated earlier, neither the BLAS library, not C99, nor.. give accuracies.

Synthesis of requirements

From the previous needs, what are the requirements applicable to SONNX? The following list is an unsorted set of potential requirements. Some of them may be applicable to a specific (sub-)profile.

- [**REQ**] The MLMD shall specify the the exact ordering of operations, the representation of numbers, the roundings.
- [**REQ**] The MLMD shall specify the accuracy of all operations
- [**REQ**] The MLMD shall specify the exact input domain in which the model shall provide an output.
- [**REQ**] The SONNX standard shall specify th eexpected behaviour should a runtim error occur (over/under-flow, division by zero,...).
- [**REQ**] The SONNX standard shall provide exactly rounded operators.

- ...

Solutions

What we can / cannot do...

- Specifying the error seems not possible...
- Describing how operations are done seems to be the obly way to ensure replicability of results...
- Could we provide a test suite? (qualification kit as for ESA's MLFS)
- ** Could we provide a multiprecision implementation using e.g., `mpfr` [mpfr]?

Means of analysis techniques and tools

- What are the technical means available to estimate the impact of errors on results? (e.g., fluctuat, CADNA...)
- See [Beu-24, Ch. 4] who proposes forward / backward error estimation for neural networks, taking into account nn linear activation functions.
- Do we need the intervention of some FP experts? Who?

References

- Deterministic and probabilistic backward error analysis of neural networks in floating-point arithmetic
- [Jia-XX] Kai Jia and Martin Rinard, Exploiting Verified Neural Networks via Floating point Numerical Errors, here
- [Beu-24] Théo Beuzeville, Analyse inverse des erreurs des réseaux de neurones artificiels avec applications aux calculs en virgule flottante et aux attaques adverses
- [Gu-13] Eric Goubault. “Static Analysis by Abstract Interpretation of Numerical Programs and Systems, and FLUCTUAT”. In: Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings. Ed. by Francesco Logozzo and Manuel Fähndrich. Vol. 7935. Lecture Notes in Computer Science. Springer, 2013, pp. 1–3.
- [Iou-19] Arnault Ioualalen and Matthieu Martel. “Neural Network Precision Tuning”. In: Quantitative Evaluation of Systems, 16th International Conference, QEST 2019, Glasgow, UK, September 10-12, 2019, Proceedings. Ed. by David Parker and Verena Wolf. Vol. 11785. Lecture Notes in Computer Science. Springer, 2019, pp. 129–143.
- [Sin-18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. “Fast and Effective Robustness Certification”. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 10825–10836.
- [ieee754] IEEE Standard for Floating-Point Arithmetic, IEEE Computer Society, IEEE 754-2019, 2019/06/13
- [Mic-22] P. Micikeviciuset al., "FP8 formats for deep learning," arXiv (Cornell University), Sep. 2022, doi: 10.48550/arxiv.2209.05433
- [Dar-06] C. Daramy-Loirat and D. Defour, ‘CR-LIBM A library of correctly rounded elementary functions in double-precision’. Dec. 2006. Available online

- [Sch-18] Fabian Schriever, Software User-Manual - Basic mathematical Library for Flight Software, E1356-GTD-SUM01, 2018/05/23
- [mpfr] GNU MPFR - The Multiple Precision Floating-Point Reliable Library, August 2013, Available online
- [Hou-07] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, 'MPFR: A multiple-precision binary floating-point library with correct rounding', ACM Trans. Math. Softw., vol. 33, no. 2, p. 13, Jun. 2007, doi: 10.1145/1236463.1236468.
- [Dyd-23] Anton Rydahl, Joseph Huber, Ethan uis Mcdonnough, Johannes Doerfert, "Precision and Performance Analysis of C Standard Math Library Functions on GPUs", Available online
- [Gla-24] B. Gladman, V. Innocente, J. Mather, and P. Zimmermann, 'Accuracy of Mathematical Functions in Single, Double, Double Extended, and Quadruple Precision'. Available online
- [Cho-16] Chohra, C., Langlois, P., Parello, D. (2017). Reproducible, Accurately Rounded and Efficient BLAS. In: Desprez, F., et al. Euro-Par 2016: Parallel Processing Workshops. Euro-Par 2016. Lecture Notes in Computer Science, vol 10104. Springer, Cham. https://doi.org/10.1007/978-3-319-58943-5_49
- [Dem-15] J. Demmel and H. D. Nguyen, 'Parallel Reproducible Summation', IEEE Transactions on Computers, vol. 64, no. 7, pp. 2060–2070, Jul. 2015, doi: 10.1109/TC.2014.2345391.
- [Col-15] C. Collange, D. Defour, S. Graillat, and R. Iakymchuk, 'Numerical reproducibility for the parallel reduction on multi- and many-core architectures', Parallel Computing, vol. 49, pp. 83–97, Nov. 2015, doi: 10.1016/j.parco.2015.09.001.
- [Bru-15] N. Brunie, F. De Dinechin, O. Kupriianova, and C. Lauter, 'Code Generators for Mathematical Functions', in 2015 IEEE 22nd Symposium on Computer Arithmetic, Lyon: IEEE, Jun. 2015, pp. 66–73. doi: 10.1109/ARITH.2015.22. Available online
- [ISO26262] ISO, "Road vehicles — Functional safety — Part 6: Product development at the software level", ISO 26262-6
- [DO178C] RTCA, "Software Considerations in Airborne Systems and Equipment Certification", DO-178C
- [ARP6983] SAE International, "Recommended Practice for Development and Certification/Approval of Aeronautical Safety-Related Products Implementing ML", preliminary version.
- [Del] David Delmas, Eric Goubault, Sylve Putot, Jean Souyris, Karim Tekkal, Franck Védrine. "Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software." Available online.
- [Mul-10] J.-M. Muller et al., Handbook of Floating-Point Arithmetic. Birkhäuser Verlag, 2010.
- [Mur-19] A. Murthy, H. Das, and M. A. Islam, 'Robustness of Neural Networks to Parameter Quantization', Mar. 26, 2019, arXiv: arXiv:1903.10672. Accessed: Nov. 06, 2024. Online
- [Sch-23] A. Schlögl, N. Hofer, and R. Böhme, 'Causes and Effects of Unanticipated Numerical Deviations in Neural Network Inference Frameworks', Neurips 2023, Available online

Attic

- Do we really need to add the replication in the SONNX standard? I would say "yes" in the sense that the replication criteria (which I would call "implementation criteria" since these criteria allow discriminating a correct implementation from an incorrect one)
- Check if formal methods (e.g., *fluctuat*) could be used.
- Be careful of the input domain of variables...
- The replication criterion can mention a reference implementation, otherwise, the replication refers to the formal specification given by the model itself (see above)
- In practice, what are the verifications performed on the ML model that will not be performed on the implementation of the model? What shall the ML model contain to be able to preserve these properties?

- Do we have to care about *training* reproducibility?
- Up to what level of requirements shall we go considering (i) the type of algorithms (that are inherently erroneous in th general case) and (ii) the assurance level targeted (DAL C in aeronautics)?