INTERNATIONAL STANDARD

ISO/IEC 24029-2

First edition
2023-08

Artificial intelligence (AI) — Assessment of the robustness of neural networks —

Part 2: Methodology for the use of formal methods

Intelligence artificielle (IA) — Evaluation de la robustesse de réseaux neuronaux —

Partie 2: Méthodologie pour l'utilisation de méthodes formelles

Reference number
ISO/IEC 24029-2:2023(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 42, *Artificial intelligence,* in collaboration with the European Committee for Standardization (CEN) Technical Committee CEN/CLC/JTC 21, *Artificial Intelligence,* in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement).

A list of all parts in the ISO/IEC 24029 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Neural networks are widely used to perform complex tasks in various contexts, such as image or natural language processing and predictive maintenance. AI system quality models comprise certain characteristics, including robustness. For example, ISO/IEC 25059:2023,[1] which extends the SQuaRE International Standards[2] to AI systems, considers in its quality model that robustness is a sub-characteristic of reliability. Demonstrating the ability of a system to maintain its level of performance under varying conditions can be done using statistical analysis, but proving it requires some form of formal analysis. In that regard formal methods can be complementary to other methods in order to increase trust in the robustness of the neural network.

Formal methods are mathematical techniques for rigorous specification and verification of software and hardware systems with the goal to prove their correctness. Formal methods can be used to formally reason about neural networks and prove whether they satisfy relevant robustness properties. For example, consider a neural network classifier that takes as input an image and outputs a label from a fixed set of classes (such as car or airplane). Such a classifier can be formalized as a mathematical function that takes the pixel intensities of an image as input, computes the probabilities for each possible class from the fixed set, and returns a label corresponding to the highest probability. This formal model can then be used to mathematically reason about the neural network when the input image is modified. For example, suppose when given a concrete image for which the neural network outputs the label "car" the following question can be asked: "does the network output a different label if the value of an arbitrary pixel in the image is modified?" This question can be formulated as a formal mathematical statement that is either true of false for a given neural network and image.

A classical approach to using formal methods consists of three main steps that are described in this document. First, the system to be analyzed is formally defined in a model that precisely captures all possible behaviours of the system. Then, a requirement is mathematically defined. Finally, a formal method, such as solver, abstract interpretation or model checking, is used to assess whether the system meets the given requirement, yielding either a proof, a counterexample or an inconclusive result.

This document covers several available formal method techniques. At each stage of the life cycle, the document presents criteria that are applicable to assess the robustness of neural networks and to establish how neural networks are verified by formal methods. Formal methods can have issues in terms of scalability, however, they are still applicable to all types of neural networks performing various tasks on several data types. While formal methods have long been used on traditional software systems, the use of formal methods on neural networks is fairly recent and is still an active field of investigation.

This document is aimed at helping AI developers who use neural networks and who are tasked with assessing their robustness throughout the appropriate stages of the AI life cycle. ISO/IEC TR 24029-1 provides a more detailed overview of the techniques available to assess the robustness of neural networks, beyond the formal methods described in this document.

Artificial intelligence (AI) — Assessment of the robustness of neural networks —

Part 2: Methodology for the use of formal methods

1 Scope

This document provides methodology for the use of formal methods to assess robustness properties of neural networks. The document focuses on how to select, apply and manage formal methods to prove robustness properties.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 22989:2022, *Information technology — Artificial intelligence — Artificial intelligence concepts and terminology*

ISO/IEC 23053:2022, *Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 22989:2022, ISO/IEC 23053:2022 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
domain
set of possible inputs to a neural network characterized by attributes of the environment

EXAMPLE 1 A neural network performing a natural language processing task is manipulating texts composed of words. Even though the number of possible different texts is unbounded, the maximum length of each sentence is always bounded. An attribute describing this domain can therefore be the maximum length allowed for each sentence.

EXAMPLE 2 A face capture domain requirements can rely on attributes such as that the size of faces is at least 40 pixels by 40 pixels. That half-profile faces are detectable at a lower level of accuracy, provided most of the facial features are still visible. Similarly, partial occlusions are handled to some extent. Detection typically requires that more than 70 % of the face is visible. Views where the camera is the same height as the face perform best and performance degrades as the view moves above 30 degrees or below 20 degrees from straight on.

Note 1 to entry: An attribute is used to describe a bounded object even though the domain can be unbounded.

3.2
attribute
property or characteristic of an object that can be distinguished quantitatively or qualitatively by human or automated means

[SOURCE: ISO/IEC/IEEE 15939:2017, 3.2, modified — "entity" replaced with "object".]

3.3
bounded domain
set containing a finite number of objects

EXAMPLE 1 The domain of all valid 8-bit RGB images with n-pixels is bounded by its size which is at most $256^{3 \times n}$.

EXAMPLE 2 The number of all valid English sentences is infinite, therefore this domain is unbounded.

Note 1 to entry: The number of objects in an unbounded domain is infinite.

3.4
bounded object
object represented by a finite number of attributes

Note 1 to entry: Contrary to a bounded object, an unbounded object is represented with an infinite number of attributes.

3.5
stability
extent to which the output of a neural network remains the same when its inputs are changed

Note 1 to entry: A more stable neural network is less likely to change its output when input changes are noise.

3.6
sensitivity
extent to which the output of a neural network varies when its inputs are changed

Note 1 to entry: A more sensitive neural network is less likely to change its outputs when input changes are informative.

3.7
architecture
fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

3.8
relevance
ordered relative importance of an input's impact on the output of a neural network as compared to all other inputs

3.9
criterion
rule on which a judgment or decision can be based, or by which a product, service, result, or process can be evaluated

[SOURCE: ISO/IEC/IEEE 15289:2019 3.1.6]

3.10
time series
sequence of values sampled at successive points in time

[SOURCE: ISO/IEC 19794-1:2011, 3.54]

3.11
reachability
property describing whether a set of states is possible to be reached by an AI agent in a given environment

3.12
piecewise linear neural network
neural network using piecewise linear activation functions

Note 1 to entry: Examples of linear activation functions are Rectify linear unit or MaxOut.

3.13
binarized neural network
neural network having parameters that are primarily binary

3.14
recurrent neural network
neural network maintaining an internal state which encodes what the neural network has learned after processing a subsequence of the input data

3.15
transformer neural network
transformer
neural network using a self-attention mechanism to weight the effect of different parts of the input data during processing

3.16
model checking
formal expression of a theory

3.17
structural-based testing
glass-box testing
white-box testing
structural testing
dynamic testing in which the tests are derived from an examination of the structure of the test item

Note 1 to entry: Structure-based testing is not restricted to use at component level and can be used at all levels, e.g. menu item coverage as part of a system test.

Note 2 to entry: Techniques include branch testing, decision testing, and statement testing.

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.80]

3.18
closed-box testing
specification-based testing
black-box testing
testing in which the principal test basis is the external inputs and outputs of the test item, commonly based on a specification, rather than its implementation in source code or executable software

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3,75]

4 Abbreviated terms

AI	artificial intelligence
BNN	binarized neural networks
GNN	graph neural networks
MILP	mixed-integer linear programming
MRI	magnetic resonance imaging
PLNN	piecewise linear neural networks
ReLU	rectified linear unit
RNN	recurrent neural networks
SAR	synthetic aperture radar
SMC	satisfiability modulo convex
SMT	satisfiability modulo theories

5 Robustness assessment

5.1 General

In the context of neural networks, robustness specifications typically represent different conditions that can naturally or adversarially change in the domain (see 5.2) in which the neural network is deployed.

EXAMPLE 1 Consider a neural network that processes medical images, where inputs fed to the neural network are collected with a medical device that scans patients. Taking multiple images of the same patient naturally does not produce identical images. This is because the orientation of the patient can slightly change, the lighting in the room can change, an object can be reflected or random noise can be added by image post-processing steps.

EXAMPLE 2 Consider a neural network that processes the outputs of sensors and onboard cameras of a self-driving vehicle. Due to the dynamic nature of the outside world, such as weather conditions, pollution and lighting conditions, the input to the neural network is expected to have wide variations of various attributes.

Importantly, these variations introduced by the environment are typically not expected to change the neural network's robustness. The robustness of the neural network can then be verified against changes to such environmental conditions by using relevant proxy specifications within the neural network's domain of use.

Robustness properties can be local or global.[10] It is more common to verify local robustness properties than global robustness properties, as the former are easier to specify. Local robustness properties are specified with respect to a sample input from the test dataset. For example, given an image correctly classified as a car, the local robustness property can specify that all images generated by rotating the original image within 5 degrees are also classified as a car. A drawback of verifying local robustness properties is that the guarantees are local to the provided test sample and do not extend to other samples in the dataset. In contrast, global robustness properties define guarantees that hold deterministically over all possible inputs.[11] For domains where input features have semantic meaning, for example, air traffic collision avoidance systems, the global properties can be specified by defining valid input values for the input features expected in a real-world deployment. Defining meaningful input values is more challenging in settings where the individual features have no semantic meaning. The set of robustness properties described in this clause is not exhaustive and it is possible that new robustness properties occur in the future.

5.2 Notion of domain

Most AI systems, including neural networks, are intended to operate in a particular environment where their performance characteristics can be defined and evaluated (typical metrics of evaluation can be found in ISO/IEC TR 24029-1:2021, Table 1). Robustness, being one of the key performance characteristics, is inseparable from the domain where a neural network is operating. The existence of a bounded domain is implicit in many neural network applications (e.g. image classification expects images of certain quality and in a certain format).

The agent paradigm shown in Figure 1 (reproduced from ISO/IEC 22989:2022, Figure 1) postulates that an agent senses its environment and acts on this environment towards achieving certain goals. The distinct concepts AI agent and environment are emphasized in this paradigm. The notion of domain captures the limitations of current technology where a neural network, being a particular type of AI agent, is technically capable of achieving its goal only if it is operating on appropriate inputs.

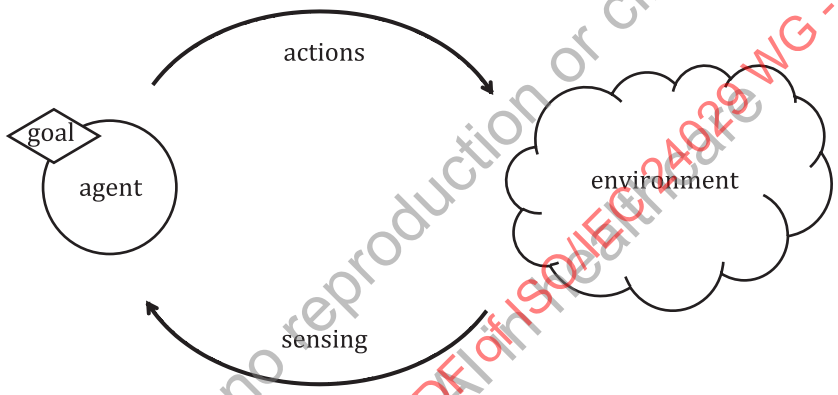

Figure 1 — The agent paradigm

The concept of domain rests on the following pillars:

— a domain shall be determined by a set of attributes which are clearly defined (i.e. the domain contains bounded objects);

— the specification of domain should be sufficient for the AI system to conduct one or more given tasks as intended;

— data used for training should be representative of data expected to be used for inference.

Establishing a domain involves specifying all data attributes essential for the neural network to be capable of achieving its goal.

Several popular domains of application of neural networks cover applications in vision, speech processing and robotics. To describe these domains, and more importantly their variability, the attributes used are generally numerical. Examples include the shape of an object in an image, the intensity of some pixels or the amplitude of an audio signal.

However, other domains can be expressed through non-numerical attributes including natural language processing, graph and Big Code (the use of automatically learning from existing code). In these cases, the attributes can be non-numerical, for example, the words in a sentence or the edges in a graph.

The attributes allow the AI developer to generate another instance in the domain from an existing instance. The attributes should be bounded in the robustness specification.

5.3 Stability

5.3.1 Stability property

A stability property expresses the extent to which a neural network output remains the same when its inputs vary over a specific domain. Checking the stability over a domain where the behaviour is supposed to hold allows for checking whether or not the performance can hold too. A stability property can be expressed either in a closed-end form (e.g. “is the variation under this threshold?”) or an open-ended form (e.g. “what is the largest stable domain?”).

In order to prove that a neural network remains performant in the presence of noisy inputs, a stability property shall be expressed. A stability property should be used on domains of uses which, in terms of expected behaviour, present some regularity properties. A stability property should not be used on a chaotic system as it is not relevant. However, even when the regularity of the domain is not easy to affirm (e.g. chaotic system), the stability property can be used to compare neural networks.

5.3.2 Stability criterion

A stability criterion establishes whether a stability property holds within a specific domain, not just for a specific set of examples or for a subset of the domain such as training or validation datasets. A stability criterion can be checked using formal methods described in 6.2.

A stability criterion shall define at least the domain value space and output value space on which it has been measured and the stability property expected.

A stability criterion may be used as one of the criteria to compare models.

For a comparison to be accurate, the following requirements shall be met:

— the neural networks perform the same task;

— the stability criterion is used on the same domain;

— the stability criterion proves the same objective.

For example, for a neural network doing classification, a stability criterion assesses whether or not a particular decision holds for every input in the domain. For a neural network doing regression, a stability criterion assesses whether or not the regression remains stable on the domain.

To be applicable, a stability criterion relies on pre-existing information of the expected output of the neural network. This information can be known by the AI developer or can be determined by another means (using simulation or solver systems). It is well-suited to assess the robustness over a domain where the expected answer is known to be similar. For this reason, a stability criterion is recommended for any decision-making process handled by a neural network (e.g. classification, identification).

5.4 Sensitivity

5.4.1 Sensitivity property

A sensitivity property on a neural network expresses the extent to which the output of a neural network varies when its inputs are changed. In order to assess the robustness on a domain, it is sometimes necessary to check the variability of a system. A sensitivity analysis can be carried out to determine how much the system varies and the inputs which can influence that variance. This analysis is then compared to a pre-existing understanding of the expected performance of the system.

When a sensitivity analysis is used to determine whether a neural network stays bounded, the sensitivity analysis shall be used over a domain. As is the case for the stability property, sensitivity analysis is more suited for domains of use which present some regularity properties.

5.4.2 Sensitivity criterion

As a sensitivity criterion expresses a property over a domain (and not a specific set of examples) it can be checked using formal methods described in 6.2.

A sensitivity criterion shall define at least the domain on which it has been measured and what are the sensitivity thresholds to be checked.

A sensitivity criterion may be used to compare different neural network architectures or trained models. For a comparison to be accurate, the following requirements shall be met:

— the neural networks shall perform the same task;

— the sensitivity criterion shall be used on the same domain;

— the sensitivity criterion shall prove the same objective.

A sensitivity criterion is especially well-suited for neural networks performing interpolation or regression tasks. For these kinds of tasks, it often allows a direct proof against a ground truth that can hold over a domain.

A sensitivity criterion is usually expressed in a closed-form as a threshold of variation over a specific domain of variation of the inputs.

5.5 Relevance

5.5.1 Relevance property

A relevance property on a neural network expresses an ordering of the impact of the inputs on the outputs. For each output, a relevance can be calculated. It expresses the individual impact of each input on the result obtained for this output. For each output the individual impact of each input can be sorted in an ordered fashion. A relevance property checks if the ordering obtained satisfies a requirement order expressed by the AI developer. A relevance property can be checked using a variety of methods to evaluate the impact of each input. Contrary to stability and sensitivity properties, a relevance property can lead to a debate between the experts in charge of its evaluation. Indeed, two neural networks can have very different relevance property results, both of which are still considered acceptable. A method for resolving conflicting results should be included in the comparison protocol. For example, a protocol can use a voting system in order to resolve the situation.

A relevance property should be used in cases where the neural network performs a task that can be done by a human. For these cases, the justification of the output of the neural network should be understood and verified. A relevance property asserts if the performance of the system can be assured for the correct reasons. If that is the case, then the robustness of the system can be justified and not just asserted. This verification can be done manually by a human operator or automatically using references that have been checked before.

5.5.2 Relevance criterion

A relevance criterion expresses a relevance property over a domain which requires demonstration of a link between each input and the outputs. For that, it requires a method able to separate the influence of each input. Formal methods relying on symbolic calculus, logical calculus or computational methods can be used to achieve such a goal. Examples of formal methods available to check a relevance criterion are provided in 6.2.

A relevance criterion should present the domain on which it has been measured and the expected results. If the expected results cannot be defined a priori, the relevance criterion should present at least the methodology to evaluate the results.

A relevance criterion may be used to compare different neural network architectures or training outputs. For a comparison to be accurate, the following requirements shall be met:

— the neural networks shall perform the same task;

— the relevance criterion shall be used on the same domain;

— the relevance criterion shall prove the same objective.

EXAMPLE For a neural network performing a classification task, a relevance criterion can be used to check if the most relevant pixels are located on a specific part of the object to be identified (e.g. the wheels in order to identify a vehicle). For a neural network performing predictive analysis of a time series, a relevance criterion can be used to check if the predicted event matches a consequential logic acceptable for the AI developer (e.g. a soon to be faulty engine can be triggered by an over-heating alarm).

A relevance criterion can be expressed on a variety of tasks, as long as the result can be analyzed by an AI developer. A relevance criterion can be used for example, on classification, detection, interpolation or regression tasks. Checking a relevance criterion can be automated or the checking can rely on human assessment to see if the result obtained is acceptable. When the checking relies on human assessment, the decision can be transferred as a new requirement to automate tests to the degree possible.

5.6 Reachability

5.6.1 Reachability property

A reachability property on a neural network expresses the multi-step performance of the network in conjunction with its operating environment. This type of property applies to systems operating in the agent paradigm as shown in Figure 1. A reachability property checks whether an AI agent can reach a set of states when using the neural network to control itself in a given environment. A reachability property can specify either a set of failure states that the AI agent shall avoid or a set of goal states that the AI agent shall reach.

Expressing this type of property requires defining an environment model that describes the effect of an AI agent's action on its next state. The environment can evolve either deterministically or stochastically. For a deterministic environment, the reachability property expresses whether or not it is possible for the AI agent to reach a particular set of states.

5.6.2 Reachability criterion

A reachability criterion expresses a reachability property over a given set of initial states. For a deterministic environment, it can be checked using methods described in 6.2.4. For a stochastic environment, the criterion expresses a probability of reaching a set of states. This probability can be determined using methods in 6.2.5.

A reachability criterion should be satisfied for a given set of initial states. The set of initial states can be specified as part of the criterion. Alternatively, formal methods can be used to determine the set of initial states for which the neural network satisfies the criterion. An advantage of using a reachability criterion to evaluate a neural network is that it provides a metric on the performance of the network in a closed-loop environment. Therefore, it can be used to express high-level safety properties that go beyond input-output properties.

For example, in the case of an aircraft collision avoidance neural network, the reachability criterion can express a requirement to avoid reaching a set of collision states given a particular environment model.

6 Applicability of formal methods on neural networks

6.1 Types of neural network concerned

6.1.1 Architectures of neural networks

6.1.1.1 General

Neural networks can be designed and built using different kinds of architectures. Formal verification techniques for neural networks depend on their architecture. Subclause 6.1.1 describes formal techniques that have been developed for the following architectures: piecewise linear neural networks, binarized neural networks, recurrent neural networks and transformer networks. While this list is not exhaustive, and new architectures and relevant formal verification techniques can emerge, this list covers a large number of current neural network architectures and the techniques that apply. More details about the techniques mentioned are available in 6.2.

6.1.1.2 Piecewise linear neural networks

PLNNs[12] do not use non-linear functions such as sigmoid or tanh. PLNNs can use linear transformations such as fully connected or convolutional layers, pooling units such as MaxPooling, and operations such as batch-normalization or dropout that preserve piecewise linearity. The majority of current neural networks are PLNNs.

Formal verification methods have been proposed that first transform a PLNN into a mathematically equivalent set of linear classifiers, and then interpret each linear classifier by the features that dominate its prediction.[13] Other verification methods view the PLNN as a global optimization problem and use a method like satisfiability modulo theories (SMT) solver. Some even have posed formal verification of robustness as a mixed integer linear program.[14] Other methods are presented in ISO/IEC TR 24029-1. Additional verification methods include Fast-Lin – Fast-Lip,[15] CROWN[16] and formal safety analysis[17].

6.1.1.3 Binarized neural networks

In binarized neural networks (BNN) all activations are binary, making these networks memory efficient and computationally efficient, enabling the use of specialized algorithms for fast binary matrix multiplication. Various embedded applications ranging from image classification to object detection have been built using such an architecture[18].

Formal verification of such BNNs has been achieved by creating an exact representation of the BNN as a Boolean formula such that all valid pairs of inputs and outputs of a given network are solutions of the Boolean formula.[19] Verification is then achieved by using methods like Boolean satisfiability and integer linear programming[18].

6.1.1.4 Recurrent neural networks

Recurrent neural networks (RNN) allow accurate and efficient processing of sequential data in many domains including speech, finance and text. At each timestep, a RNN updates its internal state based on the input at that step and the internal state from previous steps. The final output is obtained after processing the whole input in sequence.

A recurrent neural network, used as a finite classifier, can be viewed as an infinite-state machine.[20] For such an infinite state system, a finite-state automaton can be trained using automated learning techniques such as a shadow model approximating the system at hand. The shadow model can then be used to check whether the RNN meets its specification, for example, using model checking techniques. Besides model checking, abstract interpretation can be applied for proving local robustness of RNNs used in image, audio and motion sensor data classification[21].

6.1.1.5 Transformer networks

Transformer networks can be deep learning networks with an encoder-decoder architecture.[22] The transformer starts by generating representations or embeddings for each distinct part of the input through the encoder. While doing this, it uses self-attention to aggregate information from all of the other parts of the input to generate a new internal representation for the input. This step is then repeated multiple times in parallel for all parts of the input, successively generating new internal representations. The decoder operates similarly and generates one part of the output at a time. While doing this, the decoder attends to the other previously generated parts of the output and also factors in the internal representations generated by the encoder. Transformers, thus, have complex self-attention layers that pose many challenges for verification, including cross-nonlinearity and cross-position dependency. Self-attention layers are the most challenging parts for formal verification of the robustness of transformers.

In Reference [23], a method is proposed to formally verify the robustness of transformers. A transformer layer is decomposed into a number of sub-layers, such that in each sub-layer, some operations are performed on the neurons in that sub-layer. The operations that are performed fall broadly into three categories:

— linear transformations;

— unary nonlinear functions;

— operations in self-attention.

Each sub-layer is viewed as containing *n* positions in the sequence with each position containing a group of neurons. For each of these positions, the bounds are computed from the first sub-layer to the last sub-layer.

6.1.2 Neural networks input data type

6.1.2.1 General

Neural networks can be tasked to process a variety of input data types to produce several output types possible (see 6.1.1). Applications of neural networks deal with data types such as image, times series, natural language, graph or tabular. While this list is not exhaustive and new applications can emerge, this list covers a large part of what can be processed by neural networks.

However, formal methods can have limitations on which data type that can be effectively and efficiently analysed. Usually the limitations are on:

— the scalability of their computations depending on the size on the inputs (and therefore of the network);

— the nature of the input in order to model perturbations.

The scalability limitation is quite common when applying formal methods. A neural network is designed to process some input vectors at a time. However, proving mathematically a property over a whole domain (i.e. for every input vector) is intrinsically more difficult than computing the result on some points inside the domain.

The second limitation is derived from the nature of the domain represented by the inputs and the ability to model a formal proof on the domain. This limitation is dependent on the notion of attributes that are used to describe the domain (see 5.2). In some cases, the attributes are numerical, such that it can be easy to model some meaningful variation of the attributes.

6.1.2.2 Image data

Image processing capabilities is one of the reasons for the recent success of neural networks. Their ability to process several types of images (e.g. camera, MRI, radar, sonar and SAR) of various resolutions, or even video streams, has fostered wide adoption.

From a formal method point of the view, the input space to cover is defined by the dimension of the array times the number of dimensions of each pixel. In the case of large images, this can prove challenging for the many formal methods that tie a symbol to each dimension of each input.

Several attributes can be defined for images in order to express variations in the input space. For example, the lighting of the image can be expressed as a variation of intensity of the pixels. While environmental changes can be more challenging, when an analytical definition is possible it can be expressed directly on the values of the pixels, thus allowing formal methods to be applied directly. When an analytical definition is not feasible, then the variation can be expressed following an approximation of the model applied onto the image (e.g. using a mask on the image).

6.1.2.3 Time series data

Recent advances in predictive technologies illustrate the applicability of neural network to time series data in order to make predictions or classifications. Each time series is composed of several instances that record information that is (usually) of the same data type. Formal methods can be applicable on time series as long as it is possible to analyze the type of data stored at each instance. The dimension of the input is then the product of the length of each times series multiplied by the dimension of each data instance.

To be applied to time series data, formal methods require that the information at each instance can be manipulated. Handling the number of inputs can be a challenge, because the number of inputs can be arbitrary large if each instance is considered independently.

6.1.2.4 Natural language data

Natural language data types based on text and speech can be processed by neural networks. This has been demonstrated, for example, by the very large-scale deployment of smart audio devices and by language models' ability to generate easily understood text. Natural language data are often pre-processed before being passed to the neural network.

Some variations of the input data can be easy to express formally, for example, adding noise to a recording. Other variations can be much more difficult, for example, removing or adding a word in a sentence without changing its semantics, or considering different semantic in a sentence, for different dialects of the same language. The formal methods applied in this setting reason about both the preprocessing pipeline and the neural network[21].

6.1.2.5 Graph data

Graph neural networks (GNN) have been widely applied in molecular biology, fraud detection and social sciences to process graph data for a variety of tasks such as node classification, link prediction and graph classification. Several robustness properties of GNNs have been defined based on perturbing node features as well as perturbing structural information such as adding or removing edges. While feature-based perturbations are continuous and can be handled formally in a similar manner to pixel intensity variations in images, structural perturbations are discrete and therefore require the development of specialized formal techniques.

6.1.2.6 Tabular data

Many application domains such as finance, health care and logistics rely heavily on tabular data which allows these applications to combine data of various types (e.g. numerical, symbolic, textual, categorical) and express relations between the elements. Tabular data can present both a very large number of rows and sometimes a variance within each row that can be hard to anticipate.

Applying formal methods on tabular data with heterogenous data types such as the ones described in 6.1.2.2 to 6.1.2.5 can cause limitations described previously.

6.2 Types of formal methods applicable

6.2.1 General

6.2.1.1 Consideration on the types of formal methods applicable

Subclause 6.2 describes existing formal methods applicable to the assessment of the robustness of neural networks. These methods can be classified based on the following criteria:

— they can be complete or incomplete;

— they can be deterministic or non-deterministic;

— they can use glass-box or closed-box testing techniques;

— based on real or computer arithmetic.

6.2.1.2 Complete vs incomplete verifiers

Complete verifiers can provide exact answers. They either prove the robustness property or provide a counterexample demonstrating a concrete violation of the property. A limitation of complete verifiers is that they are not effective at verifying the robustness of neural networks achieving high accuracy for challenging datasets. By contrast, incomplete verifiers use abstraction techniques that scale to high accuracy neural networks. However, incomplete verifiers can fail to prove that a robustness property actually holds.

6.2.1.3 Deterministic vs non-deterministic verifiers

When a deterministic verifier proves a robustness property, then the property holds on every input within the specified input region. However, certain models such as mixture density networks or variational autoencoders, applied in diverse domains such as stock prediction, speech recognition and image generation, do not produce a deterministic output but rather produce a distribution. For such networks, formal methods can be used to either deterministically compute parameters of the output distribution that hold for all inputs (e.g. as mean or standard deviation) or to provide formal guarantees on their robustness with high probability.

6.2.1.4 Verifiers using glass-box testing (model aware) vs verifiers using closed-box testing (model unaware)

Verifiers using glass-box testing require access to the model (i.e. internal representation of the network), including architecture and learned parameters. They do not, however, require access to the training data or the algorithm used to train the neural network. In domains where the deployed model is not accessible (e.g. it is encrypted), verifiers using glass-box testing are not applicable. In such cases, verifiers using closed-box testing can be employed. Verifiers using closed-box testing only require the ability to run the model on selected inputs. This can make verifiers using closed-box testing less precise than verifiers using glass-box testing.

6.2.1.5 Real vs computer arithmetic verifiers

Most verifiers assume that the neural network computations are performed with ideal real arithmetic (i.e. with no rounding errors). Thus, the verifiers' robustness guarantees do not hold for the actual computations performed with floating-point arithmetic or for other non-standard computer arithmetic. In contrast, sound verifiers (with respect to the underlying arithmetic) consider the computer arithmetic semantics and guarantee that their output captures the neural network output possible under those semantics. In some case, verifiers can also take into account changes in the ordering of

the computations (e.g. when only IEEE 754:2019[24] correctly rounded operators are used). When IEEE 754:2019 correctly rounded operators are not used, then a verifier can approximate the rounding done on each operator.

6.2.2 Solver

Mixed-integer linear programming (MILP) solvers[25] and satisfiability modulo theories (SMT) solvers[11][26] are deterministic, glass-box and typically complete verification methods. They encode all computations of a given neural network as a collection of constraints and then use these constraints to prove robustness properties.

In cases where complete verification methods are not achievable, these methods can be incomplete. Certain non-linear activations (such as hyperbolic functions including sigmoid and tanh) are too complex to be encoded precisely. Therefore, solvers approximate them with sound abstractions. Other non-linear activations (such as ReLU) can be precisely encoded.

To prove a given robustness property, the neural network and constraints on the input are encoded as a MILP problem, which can then be used to optimize the robustness constraint. If the bounds on the robustness constraint satisfy the constraints, the property is proven. SMT solvers pose the verification problem as a constraint satisfiability question that either holds or not.

Some techniques include symbolic linear relaxation that computes tighter bounds on the neural network outputs by keeping track of relaxed dependencies across inputs and then uses directed constraint refinement (refining the output relaxation by splitting the set of initial or intermediate neurons) to verify safety properties.[27] Other techniques propose a satisfiability modulo convex (SMC)-based algorithm combined with SMC-based pre-processing to compute finite abstractions of neural network-controlled autonomous systems[28].

6.2.3 Abstract interpretation

Abstract interpretation is a general framework for analysing large and complex deterministic[29] and probabilistic[30] systems in a scalable fashion. In the context of neural networks, it is used to provide an incomplete, deterministic and glass-box testing method that can verify the robustness of large neural networks. The verification process is as follows:

— first, the provided test input and a robustness specification collectively define a region that contains all possible perturbed inputs that can be obtained by modifying the input based on the robustness specification. This region can be represented exactly or approximately using certain geometric shapes such as boxes, zonotopes and polyhedra, or as custom abstract domains for neural networks[31];

— this region is then propagated through the neural network, such that every layer is sequentially applied to the input region. The input region is transformed into an output region containing all outputs reachable from the input region. Depending on the layer, this can introduce approximations (outputs that are unreachable from the input region);

— finally, an output region captures all possible outputs of the network for input perturbations that are formed according to the robustness specifications.

There is an inherent trade-off between precision and scalability in abstract interpretation. For example, simple abstract domains such as boxes can verify neural networks with millions of neurons within seconds but are often too imprecise to verify the desired robustness properties. On the other hand, semidefinite relaxations are more precise but do not scale to large networks. Balancing this trade-off is therefore key to achieving effective verification.

6.2.4 Reachability analysis in deterministic environments

Reachability-based neural network verification techniques combine the outputs of the solvers described in 6.2.2 with techniques in reachability analysis to provide guarantees on the closed-loop performance of neural networks operating in a given environment. The first step in this analysis is to

divide the input space into many smaller regions called cells. For each cell, the solvers from 6.2.2 can be used to determine the possible control outputs of the network in the region it defines. Using this information along with the environment model, it is possible to determine an overapproximation of the range of possible next states from any given cell. By repeating this for all cells in the initial state region over multiple time steps, an overapproximation of the set of reachable states can be determined.[32] Another approach to this problem is to encode an overapproximation of the environment dynamics as constraints in a mixed-integer program and use the mixed-integer verification technique from 6.2.2 to solve for an overapproximation of the output reachable set[33].

6.2.5 Reachability analysis in non-deterministic environments

When the environment is stochastic, the solvers in 6.2.2 can be combined with techniques in probabilistic model checking to determine the probability of reaching a set of states. Similar to the technique described in 6.2.4, the input space is divided into a set of cells and each cell is passed through a solver to determine the possible neural network outputs. Probabilistic model checking determines the probability of reaching a certain set of states from a given initial state using dynamic programming.[34] By adapting this framework to work with cells rather than single input states, an overapproximated probability of reaching a set of states when using a neural network can be obtained[35].

6.2.6 Model checking

Model checking is a method to prove that a formal expression of a theory is valid under a certain interpretation. More detailed descriptions can be found in ISO/IEC/IEEE 24765:2017 and in Reference [36]. A theory is expressed by a vocabulary of symbols comprising constants, functions and predicates to build sentences that state assertions about the intended semantics of an idea. A theory can either be expressed by sentences of a predicate logic or expressed by data patterns. Neural networks are treated as algorithms designed for discovery and use of data pattern models. The data pattern model is checked against the input.

For model checking to be valid, all models shall be checked. Model checking can be used on neural networks to prove relationships among different sorts of sets which obey some relationship.

EXAMPLE 1 The 'theory of family'[37] obeys the interpretation that implements the membership of persons belonging to a family. Thus, two arbitrary persons are proven to be members of the family or not. Then the sentence 'one person is parent of the other person' is checked for all available pairs of persons.

EXAMPLE 2 Model checking has been used in Reference [38] in order to prove the existence of adversarial inputs for a neural network. The theory is the language constituted by the letters and the weights and biases description of the neural network. The interpretation is constituted by the label attached to the image of the letter. It is possible to compute a distance between every possible pair of letters in the alphabet. Then, the model can be checked in order to ensure the predicate that every distance is greater than a specific threshold fixed by the AI developer. AI developer predefined predicates are checked by means of a neural network against a theory.

6.3 Summary

The applicability of formal methods to assess the robustness of neural networks requires taking into account several aspects. On the one hand, neural network architecture has an impact since each formal method has its own strengths and weakness to process each mathematical function used in the neural network. On the other hand, the type of data used as input to the neural network can have an impact, since input variability, numerical or categorical nature and size have direct consequences in the cost of computation and ease of formal analysis. To tackle these aspects, several formal methods are available: approaches using a solver, abstract interpretation, state reachability analysis or model checking.

Currently most common architectures and data types processed by neural networks can be analysed by at least one formal method. Each method has advantages and limitations (e.g. scalability) and can address one or more criteria described in Clause 5.

7 Robustness during the life cycle

7.1 General

The life cycle of an AI system, drawn from ISO/IEC 22989:2022, is described in Figure 2 and is composed of 7 stages.

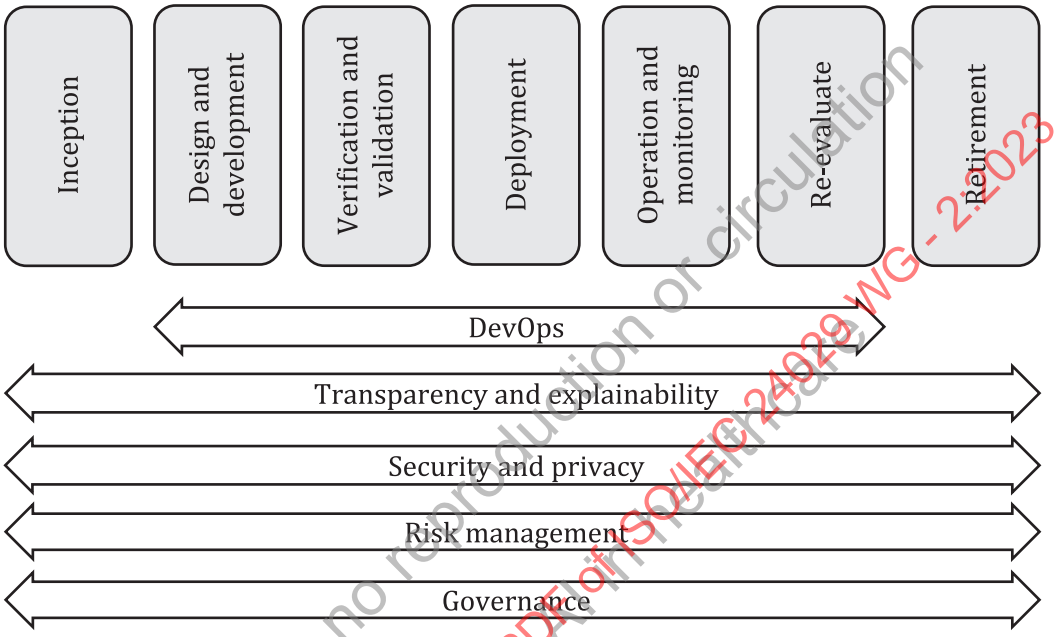

Figure 2 — Example of AI system life cycle model stages and high-level processes

ISO/IEC 22989:2022, 5.19 defines a set of AI stakeholder roles and sub-roles including AI provider, AI producer, AI developer, AI customer, etc. This clause references those roles. This clause details how to assess robustness of neural network during the design and development, verification and validation, deployment and operation monitoring.

7.2 During design and development

7.2.1 General

Even at an early stage of development, checking the robustness of a neural network can help its design. By learning early on the potential flaws in terms of robustness an AI developer can then take the necessary steps to mitigate them and in return avoid insufficient robustness later on (mitigating flaws in robustness is not covered in this document). During this step, it is assumed that the training data and the neural network architecture are still open to modifications. Formal methods are able to measure robustness but also highlight sources of loss of robustness in order to give the AI developer some important feedback. For example, formal methods can highlight features learned by a neural network for computer vision or time series processing. Formal methods can also highlight classes prone to be confused with one another by the neural network.

7.2.2 Identifying the recognized features

Identifying the features recognized by the neural network allows the AI developer to better understand and explain or interpret the behaviour of the neural network. Therefore, it is possible to better understand what the robustness of the neural network will be. Knowing which features are more easily identified allows the AI developer to understand to what extent the neural network will be able to complete its task when presented with production data.

A neural network relies on some features which it can extract from the data presented to it, whether its training has been supervised, unsupervised or through reinforcement. These features are generally not directly available to the AI developer and they are not materialized in a readable fashion in its structure. Instead, they are embedded within the mathematical model created by the training. This means that the features cannot be expressed directly in a human readable fashion. The features are mathematical artefacts expressed in a very high dimensional space.

Formal methods can use symbolic or relational approaches to establish over a domain a link through the model from the inputs to the outputs of the neural network (see 6.2 for more detail). This link allows the AI developer to know how much each input impacts each output. The learned features within the model are responsible for the strength or weakness of each individual link between an input and an output. By observing the links, it is possible to observe the consequences of the learned features, and therefore have a better understanding of their impact on neural network robustness.

In order to identify some of the learned features, the AI designer should use a relevance criterion. Confirming the result of a relevance criterion can either be done manually (through a direct confirmation) or be done automatically (through an evaluation of the correspondence against a relevance target).

— In the case of a manual confirmation, an expert directly evaluates the results of the criterion. A justification of this expert can also be added to the evaluation report in order to better understand the expert's evaluation.

— In case of an automatic confirmation, the evaluation should rely on a clear relevance target on the data. An explicit method should be defined in order to measure the level of correspondence between the relevance measured on any data and the relevance target. A threshold should be set in order to check if the level of correspondence is high enough. The relevance target should be provided.

NOTE In the confirmation process, the identity and level of competency of the person responsible can be identified for traceability or diagnostic purposes. In case of automatic confirmation, the source of the targeted relevance can also be used for traceability or diagnostic purposes.

7.2.3 Checking separability

Checking separability is a technique usable on neural networks performing classification. For these neural networks, the role of the model is to predict a class based on the input data. To do this a classification model generalizes between (and beyond) the data points it was trained on. For a classifier, the more the model is able to separate the classes, the more effective its outcome is. The robustness of such a model therefore depends on its ability to effectively separate the classes.

When designing a classifier, a sensitivity criterion can be used in order to identify which classes are more separated or less separated than others. To do so, the sensitivity analysis shall utilize domains built around data points in the test data. The spread of values of the attributes should be increased gradually in order to measure what classes are starting to overlap with each other. Starting to overlap is understood as the case where the output of a neural network on one class starts to exceed the output of another class. The process stops when all classes' outputs overlap with all other class outputs.

Separability analysis results are based on the order in which classes start to overlap and the size of the domain on which these overlaps start to occur. Based on the results of the sensitivity analysis is it possible to take action either on the training data or on the neural network architecture. The goal is to improve the separability of each class by measuring comparatively each sensitivity analysis.

7.3 During verification and validation

7.3.1 General

During the verification and validation stage, the neural network is tested in order to check if it meets its requirements and objectives. Using formal methods at this stage does not replace other means of verification and validation (such as statistical testing or field trials). However, formal methods can

bring new information on the neural network robustness within a specific domain. The main advantage that formal methods bring at this stage is to allow a more general proof of the robustness, as it is done over a domain.

7.3.2 Covering parts of the input domain

The input domain on which a neural network is intended to operate can be expressed with varying degrees of difficulty. Some are very easy to define, for example, for a neural network performing a regression task over a specific set of data all contained in certain bounds. In other cases, the definition can be more complex. For example, on image processing tasks, the input domain can be characterized by some attributes (see 5.2) but the general definition of the domain cannot be an easily defined mathematical object. When applied, formal methods are used on some form of boundary computations of the output, therefore the way the input domain is defined has a significant impact on the method.

Input domains are defined by attributes that define the space to be validated, with variation of those attributes that shall be explicitly bounded. Then, formal methods are used on domains or partitions of domains using robustness criterions described in Clause 5. It can be necessary to define a part of the domain for which validation is meaningful and brings useful information for the evaluator. Any such partitioning of the input domain should be justified. In particular, the justification should highlight why the selected criterion is able to assess the robustness on this particular partitioning of the input domain.

The point of using formal methods on part of the domain is to expand the assessment of the robustness on these parts where no validation or incomplete validation has been done. Ideally, the assessment can be over the whole domain. However, in practice this is not often feasible, whether because the domain cannot easily be defined as a whole or due to the size of the domain.

Therefore, the first step is to define the part of the domain for which validation is meaningful and can bring useful information for the evaluator.

This concept can be best explained using two different examples. One with an easily defined domain, and one with a difficult to define domain.

For the first example, consider a neural network that has been trained to interpolate the behaviour of a mathematical function taking two inputs and returning one output. The bounds defining the domain of the inputs are known, and the function the neural network is designed to mimic is always defined between these bounds. For this example, it is easy to define a partition of the input by just restricting the bounds of the inputs. On this part it is then possible to use formal methods to check the bounds of the output. As the function is well-defined over this part, it is easy to check if the neural network has sufficient robustness (using a sensitivity criterion). The whole space can then be split into several parts that can be checked separately in order to broaden the assessment of the robustness on the domain.

For the second example, consider a neural network that has been trained to classify medical images in order to classify whether one specific organ is healthy or unhealthy. The size of the images is 100 by 100, images are taken at the same distance, the angle of the organ is always centred on the image and the images all come from the same machine. The input domain is not easily defined, as the size and shape of the organ can vary from one person to another. Also, the part of the image around the organ can vary. In this scenario it is not easy to know in advance the expected behaviour on part of the input domain. Formal methods can be used to consider some parts of the domain related to a variation of a parameter understandable by the validator - for example, the volume of the organ or the brightness of the background on the image.

7.3.3 Measuring perturbation impact

By relying on the description of the domain planned for the use of a neural network, it is possible to identify types of perturbation that the neural network input can be subjected to. Each perturbation can have a different impact on the level of performance of the neural network. Their combination can also have various impacts on the robustness of the neural network. During the verification and validation phase, it is possible to assess the robustness of the system on instances of these perturbations

(combined or not). Using formal methods, it is possible to assess the robustness of the system in a more general way against these perturbations.

Perturbations can be beneficial or detrimental. They can be also intentional (e.g. in case of adversarial attack) or unintentional (e.g. in case of sensor defects or environmental changes). Perturbations can be either mathematically describable or only exemplifiable.

EXAMPLE A blur perturbation on an image can be mathematically described as a convolution of a specific kernel producing the blur applied to each pixel. However, the presence of droplets on a lens causing some image defect can only be exemplified by proposing one or several masks that artificially add droplets to an image. In the first case the equivalence is straightforward since the perturbation is a mathematical function. In the second case, the application of the function is more context dependent and can correspond to the fusion of two data into one (like mixing an image and a mask).

The process to generate a perturbed input shall be explained when its setup varies from one person to another. Applying a perturbation into an input of the neural network is seen as applying a function to the inputs in order to modify them.

To assess the robustness of a neural network against a specific perturbation using a formal method, one prerequisite is having a function describing the process of application of the perturbation. This function shall rely on at least one bounded parameter. Bounded parameters are defined by minimum and maximum values. A minimum and a maximum variation shall be set on the corresponding parameters. One or more criterion (see Clause 5) shall be expressed on the domain. Then, a formal method should be used to evaluate the criteria on the represented domain with a variety of admissible perturbations.

To assess the robustness of a neural network against several specific perturbations, the function representing each perturbation can be composed with each other. It is preferable to have a commutative composition of functions, even though it is not mandatory. The same process described in this subclause then applies to the composition of the functions.

7.4 During deployment

As neural networks are non-linear systems, they are susceptible to small changes of values in their inputs. These changes can come from numerical accuracy issues that occur during the execution of the neural network. Sources of numerical accuracy issues can be caused by:

— compilers rearranging or replacing operations (e.g. using fused multiply-add[39]);

— underlying hardware rearranging operations (e.g. to benefit from pipelining operations);

— optimization done to reduce the numerical precision (e.g. quantization, using smaller floating-point operations or fixed-point arithmetic);

— change in the rounding process;

— change in the implementation of low-level numerical operators (e.g. use of non-IEEE 754:2019[24] compliant operator, an operator with incorrect rounding or with different interpolation).

These issues should be considered when integrating a neural network on a system on which one or more of these sources of numerical issues can occur. In particular, formal methods should be used to check their impact. To do so, formal methods can be used to measure bounds of the maximum rounding error that are caused by rearranging operations or changing the underlying arithmetic. In practice, the chosen formal methods are evaluated against every criterion previously used to check if they still hold.

To address these issues the AI developer can follow these series of steps.

— First, the AI developer should verify the impact of the chosen underlying arithmetic. To do this, it is necessary to first identify for each basic operation whether its rounding errors can be statically bounded over the domain of use or not.

  — When it is possible to statically bound the rounding error of an operator, formal verifiers should take them into account in the semantics used to verify the neural network. For example,