TECHNICAL REPORT

ISO/IEC TR 29119-11

First edition
2020-11

Software and systems engineering — Software testing —

Part 11:
Guidelines on the testing of AI-based systems

Reference number
ISO/IEC TR 29119-11:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

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).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

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.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

A list of all parts in the ISO/IEC/IEEE 29119 series can be found on the ISO website.

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.

Introduction

The testing of traditional systems is well-understood, but AI-based systems, which are becoming more prevalent and critical to our daily lives, introduce new challenges. This document has been created to introduce AI-based systems and provide guidelines on how they might be tested.

Annex A provides an introduction to machine learning.

This document is primarily provided for those testers who are new to AI-based systems, but it can also be useful for more experienced testers and other stakeholders working on the development and testing of AI-based systems.

As a Technical Report, this document contains data of a different kind from that normally published as an International Standard or Technical Specification, such as data on the "state of the art".

Software and systems engineering — Software testing —

Part 11: Guidelines on the testing of AI-based systems

1 Scope

This document provides an introduction to AI-based systems. These systems are typically complex (e.g. deep neural nets), are sometimes based on big data, can be poorly specified and can be non-deterministic, which creates new challenges and opportunities for testing them.

This document explains those characteristics which are specific to AI-based systems and explains the corresponding difficulties of specifying the acceptance criteria for such systems.

This document presents the challenges of testing AI-based systems, the main challenge being the test oracle problem, whereby testers find it difficult to determine expected results for testing and therefore whether tests have passed or failed. It covers testing of these systems across the life cycle and gives guidelines on how AI-based systems in general can be tested using black-box approaches and introduces white-box testing specifically for neural networks. It describes options for the test environments and test scenarios used for testing AI-based systems.

In this document an AI-based system is a system that includes at least one AI component.

2 Normative references

There are no normative references in this document.

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

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

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

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

3.1.1
A/B testing
split-run testing
statistical testing approach that allows testers to determine which of two systems or components performs better

3.1.2
accuracy
<*machine learning* (3.1.43)> performance metric used to evaluate a *classifier* (3.1.21), which measures the proportion of *classifications* (3.1.20) *predictions* (3.1.56) that were correct

3.1.3
activation function
transfer function
<*neural network* (3.1.48)> formula associated with a node in a neural network that determines the output of the node (*activation value* (3.1.4)) from the inputs to the neuron

3.1.4
activation value
<*neural network* (3.1.48)> output of an *activation function* (3.1.3) of a node in a neural network

3.1.5
adaptability
ability of a system to react to changes in its environment in order to continue meeting both functional and non-functional requirements

3.1.6
adversarial attack
deliberate use of *adversarial examples* (3.1.7) to cause a *ML model* (3.1.46) to fail

Note 1 to entry: Typically targets ML models in the form of a *neural network* (3.1.48).

3.1.7
adversarial example
input to an *ML model* (3.1.46) created by applying small perturbations to a working example that results in the model outputting an incorrect result with high confidence

Note 1 to entry: Typically applies to ML models in the form of a *neural network* (3.1.48).

3.1.8
adversarial testing
testing approach based on the attempted creation and execution of *adversarial examples* (3.1.7) to identify defects in an *ML model* (3.1.46)

Note 1 to entry: Typically applied to ML models in the form of a *neural network* (3.1.48).

3.1.9
AI-based system
system including one or more components implementing *AI* (3.1.13)

3.1.10
AI effect
situation when a previously labelled *AI* (3.1.13) system is no longer considered to be AI as technology advances

3.1.11
AI quality metamodel
metamodel intended to ensure the quality of *AI-based systems* (3.1.9)

Note 1 to entry: This metamodel is defined in detail in DIN SPEC 92001.

3.1.12
algorithm
ML algorithm
<*machine learning* (3.1.43)> algorithm used to create an *ML model* (3.1.46) from the *training data* (3.1.80)

EXAMPLE ML algorithms include linear regression, logistic regression, *decision tree* (3.1.25), SVM, Naive Bayes, kNN, K-means and random forest.

3.1.13
artificial intelligence
AI
capability of an engineered system to acquire, process and apply knowledge and skills

3.1.14
autonomous system
system capable of working without human intervention for sustained periods

3.1.15
autonomy
ability of a system to work for sustained periods without human intervention

3.1.16
back-to-back testing
differential testing
approach to testing whereby an alternative version of the system is used as a *pseudo-oracle* (3.1.59) to generate expected results for comparison from the same test inputs

EXAMPLE The pseudo-oracle may be a system that already exists, a system developed by an independent team or a system implemented using a different programming language.

3.1.17
backward propagation
<*neural network* (3.1.48)> method used in artificial neural networks to determine the weights to be used on the network connections based on the computed error at the output of the network

Note 1 to entry: It is used to train *deep neural networks* (3.1.27).

3.1.18
benchmark suite
collection of benchmarks, where a benchmark is a set of tests used to compare the performance of alternatives

3.1.19
bias
<*machine learning* (3.1.43)> measure of the distance between the predicted value provided by the *ML model* (3.1.46) and a desired fair *prediction* (3.1.56)

3.1.20
classification
<*machine learning* (3.1.43)> machine learning function that predicts the output class for a given input

3.1.21
classifier
<*machine learning* (3.1.43)> *ML model* (3.1.46) used for *classification* (3.1.20)

3.1.22
clustering
grouping of a set of objects such that objects in the same group (i.e. a cluster) are more similar to each other than to those in other clusters

3.1.23
combinatorial testing
black-box test design technique in which test cases are designed to execute specific combinations of values of several *parameters* (3.1.53)

EXAMPLE *Pairwise testing* (3.1.52), all combinations testing, each choice testing, base choice testing.

3.1.24
confusion matrix
table used to describe the performance of a classifier (3.1.21) on a set of *test data* (3.1.75) for which the true and false values are known

3.1.25
decision tree
<*machine learning* (3.1.43)> supervised-learning *model* (3.1.46) for which inference can be represented by traversing one or more tree-like structures

3.1.26
deep learning
approach to creating rich hierarchical representations through the training of *neural networks* (3.1.48) with one or more hidden layers

Note 1 to entry: Deep learning uses multi-layered networks of simple computing units (or "neurons"). In these neural networks each unit combines a set of input values to produce an output value, which in turn is passed on to other neurons downstream.

3.1.27
deep neural net
neural network (3.1.48) with more than two layers

3.1.28
deterministic system
system which, given a particular set of inputs and starting state, will always produce the same set of outputs and final state

3.1.29
distributional shift
dataset shift
<*machine learning* (3.1.43)> distance between the *training data* (3.1.80) distribution and the desired data distribution

Note 1 to entry: The effect of distributional shift often increases as the users' interaction with the system (and so the desired data distribution) changes over time.

3.1.30
drift
degradation
staleness
<*machine learning* (3.1.43)> changes to *ML model* (3.1.46) behaviour that occur over time

Note 1 to entry: These changes typically make *predictions* (3.1.56) less accurate and may require the model to be re-trained with new data.

3.1.31
explainability
<*AI* (3.1.13)> level of understanding how the *AI-based system* (3.1.9) came up with a given result

3.1.32
exploratory testing
experience-based testing in which the tester spontaneously designs and executes tests based on the tester's existing relevant knowledge, prior exploration of the test item (including the results of previous tests), and heuristic "rules of thumb" regarding common software behaviours and types of failure

Note 1 to entry: Exploratory testing hunts for hidden properties (including hidden behaviours) that, while quite possibly benign by themselves, could interfere with other properties of the software under test, and so constitute a risk that the software will fail.

3.1.33
F1-score
<*machine learning* (3.1.43)> performance metric used to evaluate a *classifier* (3.1.21), which provides a balance (the harmonic average) between *recall* (3.1.61) and *precision* (3.1.55)

3.1.34
false negative
incorrect reporting of a failure when in reality it is a pass

Note 1 to entry: This is also known as a Type II error.

EXAMPLE The referee awards an offside when it was a goal and so reports a failure to score a goal when a goal was scored.

3.1.35
false positive
incorrect reporting of a pass when in reality it is a failure

Note 1 to entry: This is also known as a Type I error.

EXAMPLE The referee awards a goal that was offside and so should not have been awarded.

3.1.36
feature engineering
feature selection
<*machine learning* (3.1.43)> activity in which those attributes in the raw data that best represent the underlying relationships that should appear in the *model* (3.1.46) are identified for use in the *training data* (3.1.80)

3.1.37
flexibility
ability of a system to work in contexts outside its initial specification (i.e. change its behaviour according to its actual situation to satisfy its objectives)

3.1.38
fuzz testing
software testing approach in which high volumes of random (or near random) data, called fuzz, are used to generate inputs to the test item

3.1.39
general AI
strong AI
AI (3.1.13) that exhibits intelligent behaviour comparable to a human across the full range of cognitive abilities

3.1.40
graphical processing unit
GPU
application-specific integrated circuit (ASIC) specialized for display functions such as rendering images

Note 1 to entry: GPUs are designed for parallel data processing of images with a single function, but this parallel processing is also useful for executing AI-based software, such as *neural networks* (3.1.48).

3.1.41
hyperparameter
<*neural network* (3.1.48)> variable used to define the structure of a neural network and how it is trained

Note 1 to entry: Typically, hyperparameters are set by the developer of the *model* (3.1.46) and may also be referred to as a tuning *parameter* (3.1.53).

3.1.42
interpretability
<*AI* (3.1.13)> level of understanding how the underlying (AI) technology works

3.1.43
machine learning
ML
process using computational techniques to enable systems to learn from data or experience

3.1.44
metamorphic relation
description of how a change in the test inputs from the source test case to the follow-up test case affects a change (or not) in the expected outputs from the source test case to the follow-up test case

3.1.45
metamorphic testing
testing where the expected results are not based on the specification but are instead extrapolated from previous actual results

3.1.46
model
ML model
<*machine learning* (3.1.43)> output of a *ML algorithm* (3.1.12) trained with a training dataset that generates *predictions* (3.1.56) using patterns in the input data

3.1.47
narrow AI
weak AI
AI (3.1.13) focused on a single well-defined task to address a specific problem

3.1.48
neural network
artificial neural network
network of primitive processing elements connected by weighted links with adjustable weights, in which each element produces a value by applying a nonlinear function to its input values, and transmits it to other elements or presents it as an output value

Note 1 to entry: Whereas some neural networks are intended to simulate the functioning of neurons in the nervous system, most neural networks are used in *artificial intelligence* (3.1.13) as realizations of the connectionist *model* (3.1.46).

Note 2 to entry: Examples of nonlinear functions are a threshold function, a sigmoid function, and a polynomial function.

[SOURCE: ISO/IEC 2382:2015, 2120625, modified — The admitted term "neural net" has been removed; notes 3 to 5 to entry have been removed.]

3.1.49
neuron coverage
proportion of activated neurons divided by the total number of neurons in the *neural network* (3.1.48) (normally expressed as a percentage) for a set of tests

Note 1 to entry: A neuron is considered to be activated if its *activation value* (3.1.4) exceeds zero.

3.1.50
non-deterministic system
system which, given a particular set of inputs and starting state, will not always produce the same set of outputs and final state

3.1.51
overfitting
<*machine learning* (3.1.43)> generation of a *ML model* (3.1.46) that corresponds too closely to the *training data* (3.1.80), resulting in a model that finds it difficult to generalize to new data

3.1.52
pairwise testing
black-box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input *parameters* (3.1.53)

Note 1 to entry: Pairwise testing is the most popular form of *combinatorial testing* (3.1.23).

3.1.53
parameter
<*machine learning* (3.1.43)> parts of the *model* (3.1.46) that are learnt from applying the *training data* (3.1.80) to the *algorithm* (3.1.12)

EXAMPLE Learnt weights in a neural net.

Note 1 to entry: Typically, parameters are not set by the developer of the model.

3.1.54
performance metrics
<*machine learning* (3.1.43)> metrics used to evaluate *ML models* (3.1.46) that are used for *classification* (3.1.20)

EXAMPLE Typical metrics include *accuracy* (3.1.2), *precision* (3.1.55), *recall* (3.1.61) and *F1-score* (3.1.33).

3.1.55
precision
<*machine learning* (3.1.43)> performance metric used to evaluate a *classifier* (3.1.21), which measures the proportion of predicted positives that were correct

3.1.56
prediction
<*machine learning* (3.1.43)> machine learning function that results in a predicted target value for a given input

EXAMPLE Includes *classification* (3.1.20) and *regression* (3.1.62) functions.

3.1.57
pre-processing
<*machine learning* (3.1.43)> part of the ML workflow that transforms raw data into a state ready for use by the *ML algorithm* (3.1.12) to create the *ML model* (3.1.46)

Note 1 to entry: Pre-processing can include analysis, normalization, filtering, reformatting, imputation, removal of outliers and duplicates, and ensuring the completeness of the dataset.

3.1.58
probabilistic system
system whose behaviour is described in terms of probabilities, such that its outputs cannot be perfectly predicted

3.1.59
pseudo-oracle
derived test oracle
independently derived variant of the test item used to generate results, which are compared with the results of the original test item based on the same test inputs

Note 1 to entry: Pseudo-oracles are a useful alternative when traditional *test oracles* (3.1.76) are not available.

3.1.60
reasoning technique
<*AI* (3.1.13)> form of AI that generates conclusions from available information using logical techniques, such as deduction and induction

3.1.61
recall
sensitivity
<*machine learning* (3.1.43)> performance metric used to evaluate a *classifier* (3.1.21), which measures the proportion of actual positives that were predicted correctly

3.1.62
regression
<*machine learning* (3.1.43)> machine learning function that results in a numerical or continuous output value for a given input

3.1.63
regulatory standard
standard promulgated by a regulatory agency

3.1.64
reinforcement learning
<*machine learning* (3.1.43)> task of building a *ML model* (3.1.46) using a process of trial and reward to achieve an objective

Note 1 to entry: A reinforcement learning task can include the training of a ML model in a way similar to *supervised learning* (3.1.74) plus training on unlabelled inputs gathered during the operation phase of the *AI* (3.1.13) system. Each time the model makes a *prediction* (3.1.56), a reward is calculated, and further trials are run to optimize the reward.

Note 2 to entry: In reinforcement learning, the objective, or definition of success, can be defined by the system designer.

Note 3 to entry: In reinforcement learning, the reward can be a calculated number that represents how close the AI system is to achieving the objective for a given trial.

3.1.65
reward hacking
activity performed by an agent to maximise its reward function to the detriment of meeting the original objective

3.1.66
robot
programmed actuated mechanism with a degree of *autonomy* (3.1.15), moving within its environment, to perform intended tasks

Note 1 to entry: A robot includes the control system and interface of the control system.

Note 2 to entry: The *classification* (3.1.20) of robot into industrial robot or service robot is done according to its intended application.

3.1.67
safety
expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered

[SOURCE: ISO/IEC/IEEE 12207:2017, 3.1.48]

3.1.68
search algorithm
<*AI* (3.1.13)> *algorithm* (3.1.12) that systematically visits a subset of all possible states (or structures) until the goal state (or structure) is reached

3.1.69
self-learning system
adaptive system that changes its behaviour based on learning from the practice of trial and error

3.1.70
sign change coverage
proportion of neurons activated with both positive and negative *activation values* (3.1.4) divided by the total number of neurons in the *neural network* (3.1.48) (normally expressed as a percentage) for a set of tests

Note 1 to entry: An activation value of zero is considered to be a negative activation value.

3.1.71
sign-sign coverage
coverage level achieved if by changing the sign of each neuron it can be shown to individually cause one neuron in the next layer to change sign while all other neurons in the next layer stay the same (i.e. they do not change sign)

3.1.72
simulator
<testing> device, computer program or system used during testing, which behaves or operates like a given system when provided with a set of controlled inputs.

3.1.73
software agent
digital entity that perceives its environment and takes actions that maximize its chance of successfully achieving its goals

3.1.74
supervised learning
<*machine learning* (3.1.43)> task of learning a function that maps an input to an output based on labelled example input-output pairs

3.1.75
test data
<*machine learning* (3.1.43)> independent dataset used to provide an unbiased evaluation of the final, tuned *ML model* (3.1.46)

3.1.76
test oracle
source of information for determining whether a test has passed or failed

Note 1 to entry: The test oracle is often a specification used to generate expected results for individual test cases, but other sources may be used, such as comparing actual results with those of another similar program or system or asking a human expert.

3.1.77
test oracle problem
challenge of determining whether a test has passed or failed for a given set of test inputs and state

3.1.78
test scenario
situation or setting for a test item used as the basis for generating test cases

3.1.79
threshold coverage
<*neural network* (3.1.48)> proportion of neurons exceeding a threshold *activation value* (3.1.4) divided by the total number of neurons in the neural network (normally expressed as a percentage) for a set of tests

Note 1 to entry: A threshold activation value between 0 and 1 is chosen as the threshold value.

3.1.80
training data
<*machine learning* (3.1.43)> dataset used to train an *ML model* (3.1.46)

3.1.81
transparency
<*AI* (3.1.13)> level of accessibility to the *algorithm* (3.1.12) and data used by the *AI-based system* (3.1.9)

3.1.82
true negative
correct reporting of a failure when it is a failure

EXAMPLE The referee correctly awards an offside and so reports a failure to score a goal.

3.1.83
true positive
correct reporting of a pass when it is a pass

EXAMPLE The referee correctly awards a goal.

3.1.84
underfitting
<*machine learning* (3.1.43)> generation of a *ML model* (3.1.46) that does not reflect the underlying trend of the *training data* (3.1.80), resulting in a model that finds it difficult to make accurate *predictions* (3.1.56)

3.1.85
unsupervised learning
<*machine learning* (3.1.43)> task of learning a function that maps unlabelled input data to a latent representation

3.1.86
validation data
<*machine learning* (3.1.43)> dataset used to evaluate a candidate *ML model* (3.1.46) while tuning it

3.1.87
value change coverage
proportion of neurons activated where their *activation values* (3.1.4) differ by more than a change amount divided by the total number of neurons in the *neural network* (3.1.48) (normally expressed as a percentage) for a set of tests

3.1.88
virtual test environment
test environment where one or more parts are digitally simulated

3.2 Abbreviated terms

ASIC application-specific integrated circuit

API application programming interface

CEN European Committee for Standardization

CI/CD continuous integration and continuous delivery

CPU central processing unit

CENELEC European Committee for Electrotechnical Standardization

DNN deep neural network

ETSI	European Telecommunications Standards Institute
GDPR	General Data Protection Regulation
IEEE	Institute of Electrical and Electronics Engineers
IoT	internet of things
RAM	random access memory
SOTIF	safety of the intended functionality

4 Introduction to AI and testing

4.1 Overview of AI and testing

This clause introduces artificial intelligence (AI) and then explains testing in the context of AI-based systems.

Artificial intelligence is initially defined, typical AI uses cases are provided, and figures for the expanding market for AI-based systems are presented. The range of technologies used to implement AI-based systems are listed and options for the hardware and development frameworks used to implement these systems are provided. The implementation levels of narrow AI and general AI are then compared.

The importance of testing for AI-based systems is then introduced, and the use of such systems in safety-related domains is considered before the use of standards for AI-based systems is introduced.

4.2 Artificial intelligence (AI)

4.2.1 Definition of 'artificial intelligence'

To understand the term 'artificial intelligence', 'intelligence' first needs to be understood. The Oxford Dictionaries provide a suitable definition:

> *the ability to acquire and apply knowledge and skills*

Artificial intelligence (AI) is intelligence that does not occur naturally, i.e. as exhibited by humans and animals. The following definition captures this concept:

> *capability of an engineered system to acquire, process and apply knowledge and skills*

Artificial intelligence can also be considered as a discipline, leading to a second definition:

> *discipline which studies the engineering of systems with the capability to acquire, process and apply knowledge and skills (ISO/IEC 22989)*

ISO/IEC 22989[1] introduces the concepts of AI and includes a comprehensive terminology.

In practice, people's understanding of what is meant by AI changes over time – this is often known as the AI effect[2]. A strict interpretation of the above definitions may allow what we would now consider basic (non-AI) computer systems to be labelled as AI. For instance, in the 1980s an expert system based on fixed rules that performed activities traditionally carried out by bank clerks was considered to be AI, but today such systems are often considered too simple to be AI. Similarly, the Deep Blue system that beat Garry Kasparov at chess in 1997 is now derided by some as a brute force approach – and so not true AI. It is likely that today's state-of-the-art AI will also be considered 'too simple to be AI' in 20 years' time.

4.2.2 AI use cases

AI can be used for a wide variety of application areas, such as:

— Anomaly detection systems (e.g. fraud detection, health monitoring, and security)

— Autonomous systems (e.g. vehicles and trading systems)

— Computer vision systems (e.g. image classification)

— Digital assistants (e.g. Siri, Cortana)

— Email systems (e.g. spam filters)

— Intelligent speech systems (e.g. speech recognition and speech synthesis)

— Natural language processing (NLP) (e.g. deriving meaning from human language)

— Recommender systems (e.g. for purchases, films and music)

— Search engines (e.g. for searches and marketing)

— Security systems (e.g. face ID)

— Smart home devices (e.g. thermostats)

— Social media (e.g. feed personalization)

More detail on AI use cases is available in the ISO/IEC TR 24030[3]. A comprehensive list of AI use cases from a non-standard perspective can be found at Reference [4].

4.2.3 AI usage and market

AI technologies are widely used in real-world applications, such as recommending, prediction, decision making and statistical reporting. The applications are deployed in a variety of systems including autonomous driving vehicles, robot-controlled warehouses, financial forecasting applications, and security enforcement and are increasingly integrated with cloud computing, big data analytics, robotics, internet of things, mobile computing, smart cities, smart homes, intelligent healthcare, etc.

AI-based systems are becoming ever more widespread:

— The perception is that AI is the most significant technology of this time as 69 % of technology executives ranked it in the top three most significant technologies over the next 5 to 10 years[5].

— 91 % of technology executives believe AI will be at the centre of the next technological revolution[5].

— The share of jobs requiring AI skills has grown by a factor of 4.5 since 2013[6].

— Global revenues from AI for enterprise applications is projected to grow from $1.62B in 2018 to $31.2B in 2025[7].

— It is estimated that AI will add $13 trillion to the global economy over the next decade[8].

— 22 % of IT budgets are allocated to AI projects[9].

— 64 % of companies had AI projects in place or planned for next 12 months[9].

4.2.4 AI technologies

4.2.4.1 General

AI can be implemented using a wide range of approaches or technologies. These can be grouped in different ways including:

- Search algorithms
- Reasoning techniques
  - Logic programs
  - Rule engines
  - Deductive classifier
  - Case-based reasoning
  - Procedural reasoning
- Machine learning techniques (see Annex A for more detail)
  - Artificial neural networks
    - Feed forward neural networks
    - Deep learning
    - Recurrent neural networks
    - Convolutional neural networks
  - Bayesian network
  - Decision tree
  - Reinforcement learning
  - Transfer learning
  - Genetic algorithms
  - Support vector machine

Some of the most effective AI-based systems can be considered as AI hybrids, using a mix of these technologies.

ISO/IEC 22989[1] provides more details on AI concepts and on the above technologies.

4.2.4.2 Robots and software agents

Autonomous robots with electronic systems were first developed at a similar time to Alan Turing's work on machine intelligence, and robots are now widely used in factories, although the use of AI in such robots is limited[10].

A software agent is a software system that acts upon information available to it to achieve a goal. For AI, we are more often interested in intelligent software agents that are software agents capable of making decisions based on their experiences (so making them 'intelligent'). Intelligent software agents are also often labelled as autonomous as they are allowed to select which action to perform (see 4.2.4.3 for more on autonomous systems).

Intelligent software agents may work alone or with other agents to implement AI. These agents are most often located in computer systems (either physical or in the cloud) and interact with the outside world through computer interfaces. A tool using AI for performing software testing is most likely to reside in a computer system and interact with the software tester through the user interface and interact with the software it is testing through a computer interface using a defined protocol (such a tool would be considered an AI-based system as it has an AI component working with conventional, non-AI subsystems, such as the user interface). Intelligent software agents may also reside in robots; the major difference being that the robots provide the AI with a physical presence and a different way of interacting with the environment that is not available to purely computer-based software agents.

4.2.4.3 AI and autonomous systems

Autonomous systems can be physical or purely digital, and include systems for:

- Transportation
  - Cars / trucks
  - Unmanned aircraft (drones)
  - Ships / boats
  - Trains
- Robotic/IoT platforms (e.g. manufacturing, vacuum cleaners, smart thermostats)
- Medical diagnostics
- Smart buildings / smart cities / smart energy / smart utilities
- Financial systems (e.g. automated market trading systems)

The logical structure of an autonomous system can be considered as comprising three high-level functions: sensing, decision-making and control, as shown in Figure 1. Sensors (e.g. cameras, GPS, RADAR, LIDAR) provide inputs to the sensing function and are used to gather information about the system's environment, such as the positions of nearby cars, pedestrians and information on road signs for an autonomous car. Part of this 'sensing' function is also known as localization, which is determining the system's current position in the environment and relating this to maps (e.g. detailed offline maps for autonomous cars). The 'decision-making' function decides what the system's next move should be (e.g. braking, turning, climbing, descending) depending on the function provided by the autonomous system (e.g. adaptive cruise control). The 'control' function implements the decision by calling on actuators (e.g. to release air, open fuel valve).

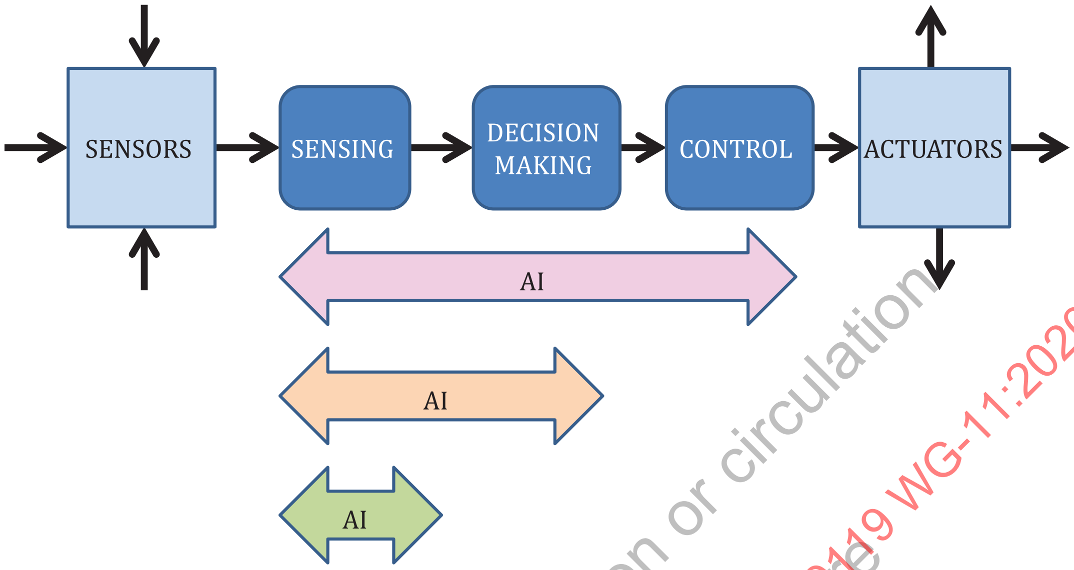

Figure 1 — Logical structure of an autonomous system

Fully autonomous systems require more, and perhaps better, sensors than their less autonomous counterparts; and to make sense of the data from these sensors, these systems typically use deep learning, a form of machine learning. To perform the necessary decision-making, the system will also often use deep learning. Thus, each of the high-level functions in the autonomous system can be implemented as AI or can be implemented using other technologies (in an autonomous car, the sensing and decision-making functions are often implemented as AI, while the control function may be implemented using conventional techniques). It is also possible to implement a complete autonomous system as a single ML system (e.g. a car steering system that learns from 'observing' manual steering based on video inputs and steering outputs).

4.2.5 AI hardware

AI-based systems, especially ML systems implemented as neural networks performing pattern recognition (e.g. machine vision, speech recognition), require many calculations to be run in parallel. General-purpose CPUs do not perform this type of calculation efficiently and, instead, graphical processing units (GPUs), which are optimised for parallel processing of images using thousands of cores are often used. GPUs are however not optimised for AI, and a new generation of hardware developed specifically for AI is now becoming available.

Many AI implementations are, by their nature, not focused on exact calculations, but rather on probabilistic determinations and so the accuracy of a 64-bit processor is often unnecessary and processors with less bits can run faster and consume less energy. Because much of the processing time and energy is involved with moving large amounts of data from RAM to the processor for relatively simple calculations, the concept of phase changing memory devices that allow simple calculations to be performed directly on memory are also being developed[11].

AI-specific hardware architectures include neural network processing units and neuromorphic computing, while existing technologies, such as field programmable gate arrays and application-specific integrated circuits can be tailored to AI workloads, as will the next generations GPUs. Some of the integrated circuits within these architectures are focused on specific areas of AI, such as image recognition. When performing machine learning (see Annex A), the processing used to train models can be quite different from the processing used to run the inferencing on the deployed model and this suggests that different processors for each activity should be considered.

4.2.6 AI development frameworks

There are several open-source AI development frameworks available, often optimised for specific application areas.

EXAMPLE The most popular AI development frameworks include:

— TensorFlow[64] – based on data flow graphs for scalable machine learning by Google

— PyTorch[65] - neural networks for deep learning in the Python language

— MxNet[66] – a deep learning open-source framework used by Amazon for AWS

— CNTK[67] – the Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit

— Keras[68] - a high-level API, written in the Python language, capable of running on top of TensorFlow or CNTK

This information is given for the convenience of users of this document and does not constitute an endorsement by ISO/IEC of the frameworks named.

4.2.7 Narrow vs general AI

Up until now, all successful AI has been 'narrow' AI, which means it can handle a single specialized task, such as playing Go, performing as a spam filter, or controlling a manoeuvre in a self-driving car.

General AI is far more advanced than narrow AI and refers to an AI-based system that can handle a number of quite disparate tasks, much the same as a human. General AI is also known as high-level machine intelligence (HLMI). A survey of AI researchers published in 2017 reported that the overall mean estimate for when HLMI would be achieved was by 2061[2]. The testing of HLMI is not within the scope of this document.

ISO/IEC 22989[1] provides coverage of AI concepts, including narrow and general AI systems.

4.3 Testing of AI-based systems

4.3.1 The importance of testing for AI-based systems

There have already been a number of widely publicized failures of AI. According to a 2019 IDC Survey, "Most organizations reported some failures among their AI projects with a quarter of them reporting up to 50 % failure rate; lack of skilled staff and unrealistic expectations were identified as the top reasons for failure."[13]

Failures have historically provided one of the most convincing drivers for performing adequate software testing. Industry surveys show a perception that AI is an important trend for software testing:

— AI was rated the number one new technology that will be important to the testing world in the next 3 to 5 years[14].

— AI was rated second (by 49.9 % of respondents) of all technologies that will be important to the software testing industry in the following 5 years[15].

— The most popular trends in software testing were AI, CI/CD, and security (equal first)[16].

However, the quality assurance of existing AI application development processes is still far from satisfactory and the demand for being able to show demonstrable levels of confidence in such systems is growing:

— 19 % of respondent are already testing AI / machine learning[14].

— 57 % of companies are experimenting with new testing approaches[9].

Software testing is a fundamental, effective and recognized quality assurance method which has shown its cost-effectiveness to ensure the reliability of many complex software systems. Moreover, the adaptation of software testing to the specifics of AI-based systems remains largely unexplored and needs extensive research to be performed[17]. Therefore, testing for AI-based systems is definitely important.

Testing is also included at a high-level as a mitigation measure for known AI vulnerabilities in the ISO/IEC TR 24028 on trustworthiness in artificial Intelligence[3].

4.3.2 Safety-related AI-based systems

AI-based systems are already beginning to be used for making decisions that affect safety and this trend will see increased use of AI for safety-related systems. Safety is defined as the 'expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered (ISO/IEC/IEEE 12207:2017).

Current standards for the assurance of safety of technical systems require a full understanding of the system under all possible conditions before its release. Many AI-based systems are probabilistic and non-deterministic (see 5.1.8) – this unpredictability makes it very difficult to make an evidence-based case that they will not cause harm. Also, the use of machine learning, such as deep learning, can result in systems that are complex (see 5.1.6) and difficult to interpret (see 5.1.7). If AI-based systems are to be used in safety-critical areas, then each of these problem areas needs to be addressed. Standards for safety-related AI-based systems are covered in 4.3.3.

4.3.3 Standardization and AI

4.3.3.1 Introduction to AI standardization

Standardization aims to promote innovation, help improve system quality, and ensure user safety, while creating a fair and open industry ecosystem. AI standardization occurs at various levels, including:

— International standards organizations

— Regional standards organizations

— National standards organizations

— Other standards organizations

Under Joint Technical Committee 1 (JTC 1) of ISO and IEC, Subcommittee 42 (SC 42) is specifically responsible for artificial intelligence standards, although AI-based systems are also considered relevant by several other ISO/IEC committees and groups, such as JTC 1/SC 7 (software and systems engineering), TC 22 (road vehicles) and ITU-T SG20 (IoT, smart cities and communities).

At the European level, ETSI and CEN-CENELEC are both involved with AI standards. ETSI has an Industry Specification Group (ISG) on Experiential Networked Intelligence (ENI), whose goal is to develop standards for a cognitive network management system incorporating a closed-loop control approach. CEN-CENELEC intends to define a standards roadmap for the AI domain that is due in 2020.

China has several AI standards initiatives at the national level, with national technical committees working on automation systems and integration (SAC/TC 159), audio, video, multimedia and equipment (SAC/TC 242) and intelligent transport systems (SAC/TC 268). SAC/TC 28 also addresses AI standardization work related to vocabulary, user interfaces and biometric feature recognition.

Germany has developed AI quality metamodel[19],[20], which is described in more detail in 4.3.3.3.

The IEEE provides a specific focus on the ethical aspects of AI-based systems. The IEEE Global Initiative for Ethical Considerations in Artificial Intelligence and Autonomous Systems has a mission "to ensure every stakeholder involved in the design and development of autonomous and intelligent systems is educated, trained, and empowered to prioritize ethical considerations so that these technologies are

advanced for the benefit of humanity." As part of this initiative, the IEEE P7000 series of standards[69] addresses specific issues at the intersection of technological and ethical considerations.

JTC 1/SC 42 are also working on the topic of ethics and AI[73].

Other standards initiatives include standards on AI tool interoperability, such as ONNX (Open Neural Network Exchange format)[70], NNEF (Neural Network Exchange Format)[71] and PMML (Predictive Model Mark-up Language)[72].

4.3.3.2 Regulatory standards for AI

4.3.3.2.1 General

Regulatory standards can be split into two broad categories: those that apply to safety-related systems and those that apply to non-safety-related systems, such as financial, utilities and reporting systems. Safety-related systems are those that could potentially cause harm to people, property or the environment. Regulatory standards often include requirements for the software testing of systems covered by these standards.

4.3.3.2.2 Non-safety-related regulatory standards

At present (in 2020), there are few international standards that apply to non-safety-related AI-based systems. However, from May 2018, the EU-wide General Data Protection Regulation (GDPR) came into effect and can cover AI-based systems. Any system that uses automated processes to make decisions with legal or similarly significant effects on individuals follows the GDPR rules that require[62] organizations using such systems to provide users with:

— specific and easily accessible information about the automated decision-making process;

— a simple way to obtain human intervention to review, and potentially change the decision.

4.3.3.2.3 Safety-related standards

AI-specific requirements for safety-related AI-based systems are currently (in 2020) poorly covered by standards and in most domains are reliant on pre-existing standards written for conventional (non-AI) systems. Some of these standards (e.g. IEC 61508[74] and ISO 26262[75]) actually specify that AI-based systems that are non-deterministic (which is many of them) should not be used for higher-integrity systems, although in practice this often means that AI-based systems are considered as special cases and follow 'tailored' versions of these standards, ignoring some of the requirements. These existing safety-related standards also require that the tools used to develop safety-related systems be suitably qualified. The currently available AI frameworks and algorithms are not qualified for use on the development of safety-related systems. Although it is possible to gain this qualification through use, the relative immaturity and rapidly evolving nature of ML algorithms would mean that it is unlikely they would satisfy current regulatory requirements in this area.

In the area of autonomous systems, which are already being used (e.g. on roads, in the air, at sea and in factories), there is a danger of a gap between practice (driven by commercial necessity) and the requirements of standards. For road vehicles a new standard, ISO/PAS 21448 on the safety of the intended functionality (SOTIF), was published in 2019. This partly bridges this gap by covering an area not covered by the existing standards that are concerned with mitigating risks due to failures. For AI-based systems, an additional problem is that they may cause harm without there being a failure – perhaps due to them simply misunderstanding the situation. SOTIF covers design, verification (e.g. requiring high coverage of scenarios) and validation (e.g. requiring use of simulations).

The U.S. Department of Transportation and the National Highway Traffic Safety Administration (NHTSA) provides guidance for the development and testing of automated driving systems in the US (Automated Driving Systems (ADS): A Vision for Safety 2.0[77]), however use of this guidance is purely voluntary.

A new standard is also being developed by UL for the safety of autonomous products in general (Standard for Safety for the Evaluation of Autonomous Products, UL 4600[78]). This standard provides assessment criteria to determine the acceptability of a safety case for the autonomous product.

4.3.3.3 The AI quality metamodel

DIN SPEC 92001-1[19] is a freely available standard that provides an AI quality metamodel intended to ensure the quality of AI-based systems. The standard defines a generic life cycle for an AI module, and assumes the use of ISO/IEC/IEEE 12207 life cycle processes[79]. Each AI module is assigned a level of risk (high or low), based on whether the AI module has relevant safety, security, privacy, or ethical attributes.

DIN SPEC 92001-2[20] is under development and describes quality requirements which are linked to the three quality pillars of functionality & performance, robustness, and comprehensibility. They also link to one or more life cycle stages and processes and they are assigned a category of model, data, platform or environment. Based on their relevance, these requirements of the AI module are classified as mandatory, highly recommended or recommended. This requirement classification and the assigned risk of the AI module are used to determine the extent to which the recommended quality requirements should be followed.

5 AI system characteristics

5.1 AI-specific characteristics

5.1.1 General

AI-based systems have both functional and non-functional requirements, the same as any system. As such, the quality characteristics in the ISO/IEC 25010 quality model, as shown in Figure 2, can be used to define, in part, the requirements of AI-based systems[21]. However, AI-based systems have some unique characteristics that are not contained with this quality model, such as flexibility, adaptability, autonomy, evolution, bias, transparency/interpretability/explainability, complexity and non-determinism. These non-functional characteristics are described in more detail in 5.1.2 to 5.1.8. The full set of quality characteristics for AI-based systems could be used as the basis for a checklist used during test planning for the identification of risks that need to be mitigated by testing. Note that there is potentially some interaction, overlap and possible conflicts between these characteristics, as there is with any set of non-functional requirements. A joint ISO/IEC project is currently underway to develop a standard in this area, titled quality model for AI-based systems[63].

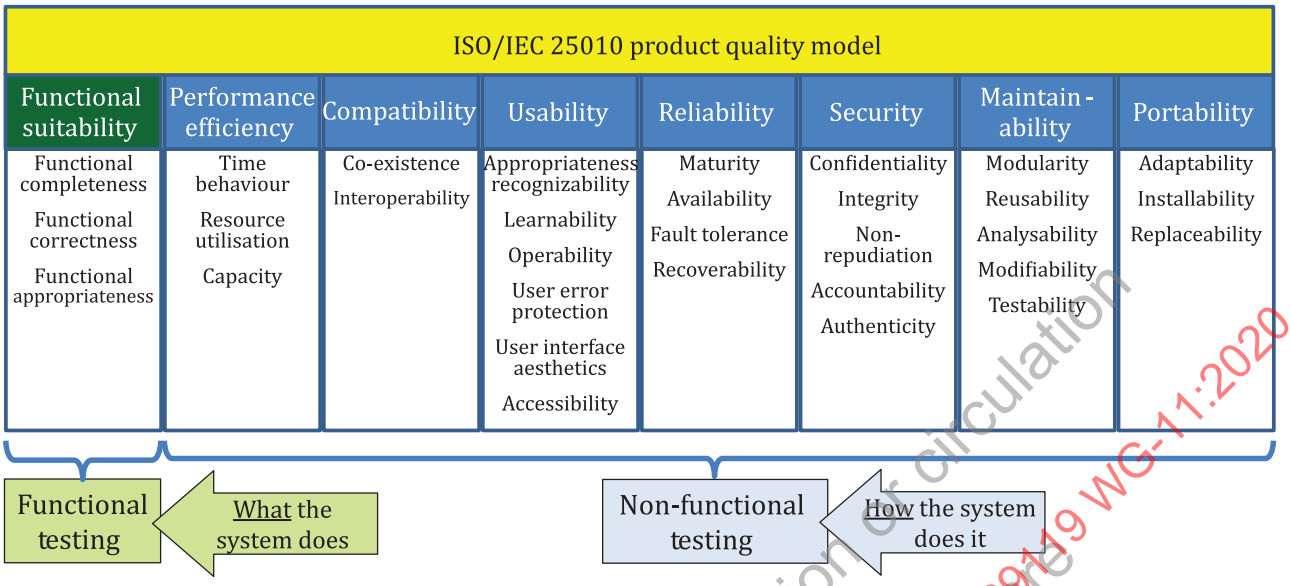

Figure 2 — ISO/IEC 25010 product quality model

5.1.2 Flexibility and adaptability

Flexibility and adaptability are closely related characteristics. Flexibility can be defined as a measure of the range of possible behaviours a system can exhibit (or states it can inhabit) – and the costs of moving between them. Adaptability can be defined as a measure of the ease with which a system can be adapted (modified)[21]. However, there are many conflicting definitions.

Both adaptability and flexibility are useful attributes of a system where the operational environment is expected to change. Such changes in the operational environment may, or may not, be specified in advance (i.e. the range of new contexts of use which the system is expected to cope with may be specified before the system is built or it may be unknown). For a system to achieve useful adaptability or flexibility it requires the ability to determine when it needs to change. Adaptative and flexible systems need to actively or passively gather information about their operational environment. Exploration (active gathering of information) provides useful information for self-improvement, but it can also be dangerous (e.g. pushing the boundaries of a flight envelope) and systems should exhibit caution when exploring in safety-related situations.

Some people believe that flexibility is one approach to achieving adaptability, with adaptation including the addition, removal, replacement or changing (flexing) of parts of the system. Others believe that adaptation is one approach to achieving flexibility (e.g. “flexibility can be achieved using different technical mechanisms, such as reactivity, pro-activity, interaction, adaptation or self-learning[22]”). Self-learning AI-based systems could be considered to be both flexible and adaptive.

Flexibility and adaptability requirements should specify those environment changes to which the system should be able to respond and also include requirements on the response process itself, such as maximum time to change, where appropriate. However, these requirements are likely to become less specific for systems where all possible future contexts of use have not been defined in detail.

5.1.3 Autonomy

Autonomy is the ability of the system to work for sustained periods without human intervention. The expected level of human intervention should be specified for the system – and so should be part of the system’s functional requirements (e.g. ‘the system will maintain cruise condition until one of the following occurs…’). Autonomy can also be considered in combination with adaptability or flexibility (e.g. system should be able to maintain a given level of adaptability or flexibility without human intervention). In some circumstances, an AI-based system may exhibit too much autonomy, in which case it may be necessary for a human to take control away from it.

5.1.4 Evolution

Evolution is when the system changes its behaviour over time. For AI-based systems, we are concerned with two forms of change. The first is when the system changes its behaviour, typically due to the system learning new (hopefully improved) behaviours as it is used (self-learning). The second type of change is when the usage profile changes and usage 'drifts' away from the original planned usage. AI-based systems are typically concerned with concept drift (the change in understanding of data and its nature over time, potentially requiring re-labelling of data) and data drift (when the data evolves with time potentially introducing previously unseen variety of data and new categories thereof). For more details see A.5.3 on distributional shift. Changes in system behaviour are not always positive, and the negative form of this system characteristic is often known as drift, degradation or staleness.

5.1.5 Bias

Bias is a measure of the distance between the predicted value provided by the machine learning (ML) model and a desired fair prediction. An AI-based system that demonstrates systematic discrimination against an individual or group of individuals is considered to be showing unfair bias. Some application areas, such as in lending, are bound by legal requirements on fairness. Bias is normally caused by the machine learning picking up unwanted patterns in the training data, such as an historic pattern of bias towards male job applicants. Training data can be compromised by both explicit and implicit bias. Implicit bias is created unintentionally, when unknown unwanted patterns in the training data exist. Explicit bias is created when known unwanted patterns in training data influence the derived model. Bias in training data can be caused by several practices, such as prejudiced labelling, historic bias and uneven sampling.

Data features that would lead to unfairness in the resultant model are either not included or handled carefully. For instance, among others, the following features can potentially cause unwanted bias:

— Gender

— Sexual orientation

— Age

— Race

— Religion

— Country of origin

— Educational background

— Source of income

— Home address

Simply removing the above features from the training data does not necessarily solve the bias problem as there could well be other features (perhaps used in combination) that could still lead to an unfair model (e.g. whether parents were divorced can lead to racial stereotyping in some locations[23]).

JTC 1/SC 42 are also working on the topic of bias in AI-based systems[80].

5.1.6 Complexity

AI-based systems, and especially those implemented through deep learning, can be extremely complex. To put this complexity in context, a typical neural network with satisfactory performance may have around 100 million parameters that were learned during training that contribute to a single decision (there are no visible 'if X and Y then result is Z' rules as found in traditional expert systems). AI-based systems may also be used for problems where there is no alternative, due to the complex nature of the problem (e.g. making decisions based on big data).

5.1.7 Transparency, interpretability and explainability

The complexity of AI-based systems (e.g. deep neural nets providing a 'black-box' implementation of AI) can lead to problems with understanding for both users and developers. This 'understanding' can generally be considered in terms of a system's transparency, interpretability and explainability, where:

— transparency – level of accessibility to the algorithm and data used by the AI-based system;

— interpretability – level of understanding how the underlying technology works;

— explainability – level of understanding how the AI-based system came up with a given result.

Different stakeholders will have different requirements for transparency, interpretability and explainability, for instance[24]:

— giving users confidence that an AI system works well;

— safeguarding against bias;

— adhering to regulatory standards or policy requirements;

— helping developers understand why a system works a certain way, assess its vulnerabilities, or verify its outputs; or

— meeting society's expectations about how individuals are afforded agency in a decision-making process. The General Data Protection Regulation (GDPR) includes requirements for explainability for certain decision-making systems (i.e. the system must provide meaningful explanations of decisions made).

The required levels of transparency, interpretability and explainability change from system to system. For instance, the results used to direct a marketing campaign are likely to need less explainability than the results for more critical systems, such as those used to support decisions on surgery or advise on jail terms (e.g. in regulated domains). For such critical systems we need explainability at least until we learn to trust the system.

There are a number of options for addressing transparency, interpretability and explainability in AI-based systems. For instance, transparency can be partially addressed by publishing details of the choice of framework, training algorithm and training data used to create the (opaque) deployed model (see Annex A for more details on this). Interpretability may be addressed by selecting models that humans find easier to understand (e.g. rule-based models, instead of deep neural networks). However, as with many non-functional requirements, there are possible conflicts between characteristics – in this case achieving interpretability may need to be traded off against required accuracy. Explainability may be achieved in some systems through visualization of how different inputs affect results.

The field of explainable AI (XAI) covers ways to make AI-based systems more explainable[25][26] (but it also covers transparency and interpretability). There are two main approaches to XAI being considered. First, looking at methods for developing AI-based systems that are inherently interpretable and second, supplementing black-box AI-based systems, such as deep neural networks, with tools that provide a level of explainability.

JTC 1 SC 42 are also working on the topic of explainability in AI-based systems[3].

5.1.8 Non-determinism

A non-deterministic system is not guaranteed to produce the same outputs from the same inputs every time it runs (in contrast to a deterministic system). With a non-deterministic system there may be multiple (valid) outcomes from a test with the same set of preconditions and test inputs. Determinism is normally assumed by testers - it allows tests to be re-run and the same results to be achieved – this is extremely useful when re-using tests for regression or confirmation testing. However, many AI-based systems are based on probabilistic implementations, meaning that they do not always produce the same results from the same test inputs. For instance, the calculation of the shortest route across a non-trivial

network (the travelling salesman problem) is known to be too complex to calculate exactly (even by a powerful computer) and sub-optimal solutions based on an initial randomly selected path are normally considered acceptable. AI-based systems can also include other causes of non-determinism, such as concurrent processing (although these are often found in complex conventional, non-AI, systems).

5.2 Aligning AI-based systems with human values

Russell[27] points out two major problems with AI-based systems. First, the specified functionality may not be perfectly aligned with the values of the human race, which are (at best) very difficult to pin down. He gives the example of King Midas, where the requested ability to turn everything he touched into gold was imparted – exactly as requested – but then found to be not what he truly wanted. A more up-to-date example is provided by Bird and Layzell[28], who used an AI-based system using genetic algorithms to generate a design for an oscillator, which resulted in a solution that involved using system's motherboard as a radio, so that it could receive oscillating signals produced by a nearby personal computer. When we specify the required objectives of AI-based systems we need to be sure that what is requested is actually what is needed – or first ensure the system is intelligent enough to provide what we request, while also taking into account human norms.

One way for AI-based systems to learn these human norms would be through observation (this may initially simply be through monitoring limited human decisions), however great care is needed to ensure that the observed human behaviour is representative and only representative of 'good' human behaviour (probably defined as excluding both deliberately bad behaviour and irrational behaviour, even if this irrational behaviour is by 'good' humans). Consideration also needs to be given to this learning of human norms being a continuing process, as what we consider acceptable behaviour today is quite different from what was considered acceptable behaviour 20 years ago – human norms can change quite quickly.

Russell's second problem is that any sufficiently capable intelligent system will prefer to ensure its own continued existence and to acquire physical and computational resources – not for their own sake, but to succeed in its assigned task. It is recognized that a sufficiently intelligent system will disable any 'off' switch early on in its operation; simply because when it is disabled it is unable to achieve its given objectives. AI-based systems will try to fulfil their given objectives, but we need to be careful of unwanted behaviours, such as those that result in side-effects (see 5.3) or reward hacking (see 5.4).

Automation complacency is a further problem that can occur in the interaction between human users and AI-based systems. This can occur when users place too much trust in an automated system and do not pay sufficient attention to monitoring system outputs. Such inattention can cause accidents, such as have been seen when the 'driver' of a (partially) self-driving vehicle fails to override the system and take control of the vehicle, when needed.

5.3 Side-effects

Side-effects occur when an AI-based system attempts to achieve its objectives and causes (typically negative) impacts on its environment. For instance, a home cleaning robot may be tasked with cleaning the kitchen in your home and decide that 'eliminating' your new puppy will help it achieve its objective. Of course, you could explicitly require your robot to accept that the puppy has a right to be in the kitchen (and therefore not be eliminated), but as AI-based systems are used in ever more complex environments it soon becomes impracticable to explicitly specify how the robot should interact with every aspect of its operational environment. For instance, your cleaning robot would also have to be told that using a high-pressure hose to clean the kitchen was not practical due to the (side-) effect of the water on the electrical appliances and sockets.

At a high level, specified objectives for AI-based systems may need to include a caveat that minimises side-effects. For narrow AI, such side-effects may be explicitly specified, but as AI-based systems become more advanced and start working in more varied operational environments it may be more efficient to define more generic caveats, such as requiring a minimal change to the environment while achieving their objective.

5.4 Reward hacking

AI-based systems using reinforcement learning (see A.1) are based on a reward function that gives the system a higher score when the system better achieves its objectives. For instance, a home cleaning robot may have a reward function based on the amount of dirt it removes from the floor – getting higher scores when the amount of dirt removed is higher. Reward hacking occurs when the AI-based system satisfies the reward function and so gets a high score, but mis-interprets the required objective. In the example of the cleaning robot, one way for it to achieve a very high score would be for it to initially make the floor extremely dirty, so giving it the opportunity to remove more dirt – a set of activities that do not meet the spirit of the initial objective of cleaning the kitchen. In this example the floor should eventually be clean (although unnecessary energy will have been expended), but there are many examples of reward hacking where the AI-based system satisfies the reward function but does not come close to achieving the required objective (e.g. a cleaning robot with a reward function based on it being able to see less visible dirt that disables its vision system).

Limiting the system's ability to innovate, however, is not the solution. One of the attractive features of AI-based systems is that they should be able to come up with smart ways to solve problems, often in ways humans would not have considered (or perhaps even understand).

5.5 Specifying ethical requirements for AI-based systems

Ethics is defined in the Cambridge Dictionary as 'a system of accepted beliefs that control behaviour, especially such a system based on morals'. As AI-based systems have become more popular, the topic of ethics and how AI-based systems should implement them is probably the most discussed topic in AI, drawing in far more people than those involved in the technical aspects of AI.

An example of the interest in ethics in AI can be seen in MIT's Moral Machine[29]. This is a platform for gathering people's opinions on moral decisions that may be made by autonomous cars, with the aim of providing guidance to the developers of such vehicles. Between 2014 and 2018 this platform gathered 40 million ethical decisions in ten languages from millions of people in 233 countries and territories. The (ongoing) study has found that there is a broad consensus that systems should give priority to younger people, priority to people over animals and priority to saving more people (e.g. save four occupants of a car over two pedestrians). The study also found that there are significant differences in the choices made by people from different parts of the world (suggesting that autonomous cars may need to follow different ethical guidelines depending on where they are to be used).

The European Commission High-Level Expert Group on Artificial Intelligence published key guidance to promote trustworthy AI in the area of ethics in April 2019[30]. It identifies the ethical principles that should be respected in the development, deployment and use of AI systems:

— Develop, deploy and use AI systems in a way that adheres to the ethical principles of respect for human autonomy, prevention of harm, fairness and explicability. Acknowledge and address the potential tensions between these principles.

— Pay particular attention to situations involving more vulnerable groups such as children, persons with disabilities and others that have historically been disadvantaged or are at risk of exclusion, and to situations which are characterised by asymmetries of power or information, such as between employers and workers, or between businesses and consumers.

— Acknowledge that, while bringing substantial benefits to individuals and society, AI systems also pose certain risks and may have a negative impact, including impacts which may be difficult to anticipate, identify or measure (e.g. on democracy, the rule of law and distributive justice, or on the human mind itself.) Adopt adequate measures to mitigate these risks when appropriate, and proportionately to the magnitude of the risk.

6 Introduction to the testing of AI-based systems

6.1 Challenges in testing AI-based systems

6.1.1 Introduction to challenges testing AI-based systems

Most AI-based systems comprise one or more AI components (e.g. a ML model) surrounded by a considerable array of traditional software that provides the supporting infrastructure, typically made up of conventional components, such as the user interface and database. Even 'pure' AI components are implemented in software and so can suffer the same defects as any other software. Thus, when testing an AI-based system, conventional software testing approaches are still required. However, AI-based systems include a number of special attributes that can make additional testing necessary than for conventional software systems:

6.1.2 System specifications

Despite the amount of recent academic research conducted on AI (and ML in particular), there is little coverage of how best to specify the expected behaviour of AI-based systems with their special characteristics (see 5.1).

In an ideal world, complete formal specifications would be available, so allowing the creation of automated test oracles. The specifications for AI-based systems are likely to be incomplete and informal, which requires testers to determine unspecified expected results, creating a test oracle problem. This can be problematic if the testers are not fully cognizant of the required system behaviour and it is difficult to get this information from domain experts.

Examples of specification challenges include when:

— the desired output of the system is not yet known, and the system is being built to provide that output;

— the real-world inputs are at such a complexity and scale, that the behaviour of the system is difficult to predict in advance;

— the required behaviour includes comparison to human qualities, including intelligence, that are difficult to define and measure.

Another problem is that AI-based systems are often specified in terms of objectives rather than required functionality, which is a more conventional approach[31]. This is because the nature of many AI-based systems is such that the functionality provided is opaque (e.g. it is very difficult to imagine how a deep neural network functions).

Some AI-based systems have extensive operational environments (e.g. an autonomous drone) and fully defining the operational environment can be more challenging than for a typical conventional system. Note that the complexity of the operational environment normally means the test environments for these systems can be equally challenging (see Clause 10 for more details on test environments).

The specifications for ML models should contain a set of required performance metrics (see A.8) to act as acceptance criteria for the ML models. Acceptance criteria including metrics may consider false positives and negatives, recognizing that 100 % accuracy is unlikely to be achieved in many use cases. Additionally, where expert judgement is required to evaluate the response of the AI-system, acceptance criteria may consider multiple evaluations, as experts may not reach consensus.

6.1.3 Test input data

AI-based systems may depend on big data inputs and/or inputs from a large range of sources. This can mean that input data is often unstructured and provided in diverse formats. When developing AI-based systems managing this data is a specialist task of a data engineer or data scientist, but when it comes to the testing, this specialist data management task is one of several performed by the tester, often with little or no specialist training.

As with all systems, where the data being processed are regulated, there may be a requirement to anonymize or otherwise control copies of real data, for example, privacy legislation such as GDPR, US legislation such as Health Insurance Portability and Accountability Act, and India's Personal Data Protection Bill. Where necessary, a sufficient level of sanitization can prevent the AI-based system under test from inferring personal details that are only partially hidden.

Sanitization of data can include de-identification of the data for privacy reasons as described in ISO/IEC 20889:2018[81].

6.1.4 Self-learning systems

As AI technology becomes more advanced, more AI-based systems will become available that can change their own behaviour over time. These may be self-adapting systems (able to reconfigure and optimize themselves) or full self-learning systems that can adapt themselves by learning from their past experiences. For both situations, it is likely that some tests that ran successfully on the original system will no longer be viable on the new, improved system. Although it may be relatively easy to identify which tests are no longer valid, it is far more difficult to ensure that new tests for the new functionality have been generated.

Another potential problem with self-learning systems is that the systems can inadvertently learn unwanted new behaviours from the testing.

6.1.5 Flexibility and adaptability

The testing of the flexibility and adaptability of an AI-based system is typically based on observing how the system changes in response to environment modification or mutation. The system's functional and non-functional requirements should be tested, and a form of regression testing, ideally automated, is often a suitable approach. The change process performed by the system should also be tested, to determine, for instance, whether the system can change within a required timeframe and whether the system stays within constraints for the resources consumed to achieve the change.

6.1.6 Autonomy

An approach to testing the autonomous behaviour of an AI-based system is to try and force the system out of its autonomous behaviour and get it to request intervention in unspecified circumstances (a form of negative testing). Negative testing can also be used to try and 'fool' the autonomous system into thinking it is in control when it should request intervention (e.g. by creating test scenarios at the boundary of its operational envelope – suggesting the application of boundary value concepts to scenario testing).

6.1.7 Evolution

Testing for system evolution (or drift) in an AI-based system normally takes the form of maintenance testing, which needs to be run on a frequent basis. This testing typically needs to monitor specified system goals, such as performance goals (e.g. accuracy, precision and sensitivity), and ensure that no data bias has been introduced to the system (e.g. Microsoft Tay chatbot[32]). The result of this testing may be that the system is re-trained, perhaps with an updated training dataset.

6.1.8 Bias

Testing for bias of an AI-based system can be performed at two stages. First, bias can be detected (and subsequently removed) in the training data through reviews, but this requires expert reviewers who can identify possible features that create bias. Second, a system can be tested for bias by the use of independent testing using bias-free testing sets. When we know that training data is biased, it may be possible to remove the source of the bias (e.g. we could remove all information that provided clues to the sex or race of the subjects). Alternatively, we could accept that a system includes bias (either implicit or explicit) but provide transparency by publishing the training data.

6.1.9 Transparency, interpretability and explainability

Testing the transparency of an AI-based system is largely concerned with determining whether there is access to the algorithm and data it uses – and can be done through review (of the documentation and referenced material, such as datasets).

Testing the interpretability of an AI-based system will be dependent on the audience as different stakeholders will hold varying levels of understanding of the underlying technology implemented in the system.

Testing the explainability of an AI-based system ideally requires the target audience (or a representative set of testers) to perform the testing to determine how easy it is for them to understand how the system comes up with a range of results.

6.1.10 Complexity

The complexity of many AI-based systems creates a test oracle problem; it may require several experts some time and discussion to agree on a single test case result from a complex AI-based system and, ideally, we would want to run many tests, which becomes infeasible if we have to rely on experts to (slowly) generate expected results. A number of test techniques can be used to address the test oracle problem, including A/B testing, back-to-back testing and metamorphic testing (see Clause 8 for more details on these techniques).

6.1.11 Probabilistic and non-deterministic systems

Due to the probabilistic nature of many AI-based systems, there is not always an exact value that can be used as an expected result. For instance, when an autonomous car plots a route around a stopped bus it does not need to calculate the optimal solution, but rather a solution that works (and is safe) - and so we accept sub-optimal, but good-enough solutions.

The nature of how AI-based systems determine their route can also mean that they do not come up with the same result each time (e.g. their calculation may be based on a random seed, which results in different, but workable, routes each time). This makes such systems non-deterministic, which results in a lack of reproducibility and means that any regression tests need to have smarter expected results that take account in the variability due to the non-determinism.

In both cases, the uncertainty in actual results requires testers to derive more sophisticated expected results, perhaps including tolerances, than for conventional systems. Probabilistic AI-based systems may also require the tester to run the same test multiple times to provide a statistically significant assurance that the system is working correctly (like a Monte Carlo experiment).

6.1.12 The test oracle problem for AI-based systems

A recurring challenge when testing AI-based systems is the test oracle problem. Poor specifications, complex, probabilistic, self-learning and non-deterministic systems make the generation of expected results problematic.

Testing approaches and techniques that address the test oracle problem are described in Clause 8 on black-box testing.

6.2 Testing AI-based systems across the life cycle

6.2.1 General

This subclause briefly considers the different test levels (sometimes called test phases) across the life cycle for an AI-based system. No assumption is made about the form of life cycle (e.g. agile, waterfall, V, iterative) as these test levels should normally apply irrespective of the life cycle used. As with all testing, the selection of testing at different levels should be based on the perceived risks and the costs of testing. Typically, testing at earlier test levels (e.g. unit and integration testing) will be cheaper and

risks that can be addressed at these levels should be tested as early as possible, however, some risks (e.g. based on the characteristics of a complete system) can only be addressed by testing a complete system and so will need to be addressed at the system test level (e.g. end-to-end scenario testing).

As described in 6.1.1, AI-based systems are typically made up of conventional components and AI components; thus, this subclause focuses on such composite AI-based systems. It does not consider the details of testing AI development frameworks (see 4.2.6), but it does consider the testing of the resultant AI components (e.g. ML models). AI-based systems can often be considered in three parts: AI component, data and user interface. The AI component is often tested similarly to a reusable software component, while the user interface is tested the same as any user interface. However, the data for AI-based systems may need to be tested slightly differently, as described in 6.2.2 to 6.2.7.

For details on the use of test environments across the life cycle, see 10.1.

6.2.2 Unit/component testing

Unit/component testing for non-AI components (e.g. user interface code) should be treated the same as for traditional systems.

Unit/component testing of ML models corresponds to the evaluation and testing stages of the ML workflow (see A.2.8 and A.2.10). As with traditional unit testing by developers, it is very rare that any defects are reported at this level of testing – and the main purpose is to improve the quality of the deliverable model.

Where ML performance metrics (see A.8.1) have been set as acceptance criteria (at the model level) then the ML model will be tested against these criteria at this test level (the acceptance criteria may form part of the evaluation and tuning activity that selects a particular ML model).

Coverage at the unit test level is traditionally concerned with either requirements or code coverage (e.g. statement, branch and decision coverage). However, coverage of ML components can be measured by the representativeness of the datasets (training, validation and test) – or, for neural networks, through coverage of the networks themselves, as described in 9.2.

Where data is pre-processed, unit tests can be used to check the pre-processing (e.g. ensuring raw data is correctly scaled or normalized).

6.2.3 Integration testing

Where an AI component is part of a larger AI-based system, it will need to be integrated into that system. There are two main approaches to integrating such an AI-based component. First, and simplest, it can be treated as an embedded component that is an integral part of the overall AI-based system. Second, the AI component can be provided as a service (typically over the web, e.g. as a web service), in which case it is provided independently of the rest of the AI-based system and is called whenever its service is needed.

Integration testing should be performed to ensure the AI component is correctly integrated with the remainder of the AI-based system of which it is a part (e.g. checking interfaces and that communicated data is correctly interpreted). For instance, tests should be performed to check that the correct image file is passed to the model for object recognition and that it is in the format expected by the model. Tests should also be performed to check that the output of the model is correctly interpreted and used by the rest of the system.

6.2.4 System testing

As with traditional systems, the system testing of AI-based systems is concerned with both functional and non-functional testing. Non-functional characteristics that are tested typically include security and performance efficiency (e.g. response time). Performance efficiency may be particularly relevant if the AI component of the overall AI-based system is provided as a service rather than as an embedded component. In addition to the quality characteristics that apply to traditional systems (e.g. as defined in

ISO/IEC 25010[21]), those AI-specific characteristics (e.g. explainability) described in 5.1 should also be considered for testing at this test level.

Where ML performance metrics (see A.8) have been set as acceptance criteria (at the system level) then the AI-based system will be tested against these criteria at this test level.

6.2.5 System integration testing

System integration testing may be especially relevant for an AI-based system when the system uses large amounts of data from other systems or when the system interacts with one or more IoT devices.

6.2.6 Acceptance testing

Business acceptance criteria should be tested as part of acceptance testing. These criteria will typically be focused on whether the AI-based system meets high-level business goals, such as those based on making or saving money, rather than on technical criteria, such as accuracy of results from a model.

Where ML performance metrics (see A.8) have been set as acceptance criteria then the AI-based system will be tested against these criteria at this test level.

Humans can over-rely on technology and where AI systems include a human-in-the-loop, the quality of the combined human and automated outputs of the system under test may not be correct. In such cases, it can be important to measure the accuracy of user-approved information, such as predictions or pre-populated fields.

6.2.7 Maintenance testing

Due to the problems associated with system evolution, it is often necessary to run regular tests to ensure that the AI-based system is still meeting its original acceptance criteria (business and technical). Where these criteria are specified as performance metrics (see A.8), these tests may be automated.

When using regression testing as part of maintenance testing the probabilistic and non-deterministic nature of many AI-based systems can cause apparent test fails when the system is simply providing a different, but acceptable, result. This may mean that the expected results for regression testing may need to be smarter than those used for deterministic systems (e.g. with an included tolerance).

Care should be taken when testing operational self-learning systems to ensure that tests do not cause the system to perform unwanted learning from the testing.

7 Testing and QA of ML systems

7.1 Introduction to the testing and QA of ML systems

Machine learning systems are described in Annex A. This clause briefly identifies the quality assurance and testing opportunities directly related to ML.

7.2 Review of ML workflow

The ML workflow that is used should be documented and followed when performing ML. Deviations from the workflow described in Annex A should be justified.

7.3 Acceptance criteria

Acceptance criteria (including both functional and non-functional requirements) should be documented and justified for use on this application. Performance metrics should be included for the model. As a minimum the AI-specific characteristics (described in 5.1) should be considered and could be used as the basis of a checklist used to determine the completeness of acceptance criteria for the AI-based system.

7.4 Framework, algorithm/model and hyperparameter selection

The choice of framework, algorithm, model, settings and hyperparameters should be documented and justified.

7.5 Training data quality

ML systems are highly dependent on the training data being representative of the operational data and some ML systems have extensive operational environments (e.g. those used in autonomous vehicles). Boundary conditions are known to cause failures in all types of system (AI and non-AI) and should be included in the training data. The selection of training data in terms of the size of the dataset and characteristics such as bias, transparency and completeness should be documented and justified and confirmed by experts where the level of risk associated with the system warrants it (e.g. for critical systems).

7.6 Test data quality

The criteria for the training data apply equally to the test data, with the caveat that the test data must be as independent of the training data as possible. The level of independence should be documented and justified. Test data should be systematically selected and/or created and should also include negative tests (e.g. inputs outside the expected input range) and adversarial tests (see 7.8 for details).

7.7 Model updates

Whenever the deployed model is updated it should be re-tested to ensure it continues to satisfy the acceptance criteria, including tests against implicit requirements that may not be documented, such as testing for model degradation (e.g. the new model runs slower than the previous model). Where appropriate, A/B testing or back-to-back testing should be performed against the previous model.

7.8 Adversarial examples and testing

An adversarial example is where an extremely small change made to the input to a neural network produces an unexpected (and wrong) large change in the output (i.e. a completely different result than for the unchanged inputs) [33]. Adversarial examples were first noticed with image classifiers. By simply changing a few pixels (not visible to the human eye) it is possible to persuade the neural network to change its image classification to a very different object (and with a high degree of confidence). Note, however, that adversarial examples are not limited to image classifiers, but are a known attribute of neural networks in general and so apply to any use made of neural networks (and may also apply to other forms of ML models).

Adversarial examples are generally transferable. This means that an adversarial example that causes one neural network to fail will often cause other neural networks to fail that are trained to perform the same task. Note that these other neural networks may have been trained with different data and based on different architectures, but they are still prone to failure with the same adversarial examples.

Adversarial testing is often referred to as performing adversarial attacks. By performing these attacks and identifying vulnerabilities during testing, measures can be taken to protect against future failures and so the robustness of the neural network is improved.

Attacks can be made when training the model and then on the trained model (neural network) itself. Attacks during training can include corrupting the training data (e.g. modifying labels), adding bad data to the training set (e.g. unwanted features) and corrupting the learning algorithm. Attacks on the trained model can be white-box or black-box and involve identifying adversarial examples that will force the model to give bad results.

With white-box attacks, the attacker has full knowledge of the algorithm used to train the model and also the settings and hyperparameters used. The attacker uses this knowledge to generate adversarial examples by, for instance, making small perturbations in inputs and monitoring which ones cause large changes to the model.

With black-box attacks, the attacker has no access to the model's internal workings or knowledge of how it was trained. In this situation, the attacker initially uses the model to determine its functionality and then builds a 'duplicate' model that provides the same functionality. The attacker then uses a white-box approach to identify adversarial examples for this duplicate model. As adversarial examples are generally transferable, the same adversarial examples will normally also work on the (black-box) model.

7.9 Benchmarks for machine learning

Ideally experts would be used to evaluate each new ML system, but that's normally too expensive. Instead, "representative" industry-standard benchmark suites are available, which include diverse workloads to cover a wide range of situations (e.g. image classification, object detection, translation and recommendation).

These benchmark suites can be used to measure the performance of both hardware (using defined models) and software (e.g. to determine the fastest models). Software benchmark suites can measure training (e.g. how fast a framework can train a ML model using a defined training dataset to a specified target quality metric, such as 75 % accuracy) and inference (e.g. how fast a trained ML model can perform inference).

Examples of ML sets of benchmarks are provided by MLPerf[34], which provides benchmarks for software frameworks, hardware accelerators and ML cloud platforms, and DAWNBench[35], which is a benchmark suite from Stanford University. The OAEI (Ontology Alignment Evaluation Initiative) is a coordinated international initiative[36] with the goals of:

— assessing strengths and weaknesses of alignment/matching systems;

— comparing performance of techniques;

— increasing communication among algorithm developers;

— improving evaluation techniques;

— helping to improve the work on ontology alignment/matching.

These goals are achieved through the controlled experimental evaluation of the techniques' performances.

8 Black-box testing of AI-based systems

8.1 Combinatorial testing

To prove, by dynamic testing, that a specific test item meets all requirements under all given circumstances, then all possible combinations of input values in all possible states would need to be tested. This impractical activity is referred to as 'exhaustive testing'. For that reason, in practice software testing derives test suites by sampling from the (extremely large) set of possible input values and states. Combinatorial testing is one systematic (and effective) approach to deriving a useful subset of combinations from this input space[37].

The combinations of interest are defined in terms of parameters (i.e. inputs and environment conditions) and the values these parameters can take. Where numerous parameters (each with numerous discrete values) can be combined, this technique enables a significant reduction in the number of test cases required, ideally without compromising the defect detection ability of the test suite.

ISO/IEC/IEEE 29119-4[38] defines several combinatorial testing techniques, such as all combinations, each choice testing, base choice testing and pairwise testing. In practice pairwise testing is the most widely used, mainly due to ease of understanding, ample tool support and research showing that most defects are caused by interactions involving few parameters[37].

The number of parameters of interest for an AI-based system can be extremely high, especially when the system uses big data or interacts with the outside world, such as a self-driving car. Thus, a means

of systematically reducing the almost infinite number of possible combinations to a manageable subset by using a combinatorial testing technique, such as pairwise testing, is extremely useful. In practice, even the use of pairwise testing can still result in extensive test suites for such systems and the use of automation and virtual test environments (see 10.1) often becomes necessary.

Using self-driving cars as an example, at a high level the scenarios for system testing need to consider both different vehicle functions and the environments in which they are expected to operate. Thus, the parameters would need to include the various self-driving functions (e.g. cruise control, adaptive cruise control, lane keeping assistance, lane change assistance, traffic light assistance, etc.) along with environment constraints (e.g. road types and surfaces, geographic area, time of day, weather conditions, traffic conditions, visibility, etc.). In addition to these parameters, inputs from sensors should be considered at varying levels of effectiveness (e.g. the input from a video camera will degrade as a journey progress and it gets dirtier or the accuracy of a GPS unit will change as different numbers of satellites come into and go out of line of sight). Research is currently unclear on the necessary level of rigour that would be required for the use of combinatorial testing with safety-critical AI-based systems such as self-driving cars (e.g. pairwise may not be sufficient), but it is known that the approach is effective at finding defects and can also be used to estimate the residual level of risk.

8.2 Back-to-back testing

In back-to-back testing, an alternative version of the system (e.g. already existing, developed by a different team or implemented using a different programming language) is used as a pseudo-oracle to generate expected results for comparison from the same test inputs. This is sometimes known as differential testing.

As such, back-to-back testing is not a test case generation technique as test inputs are not generated. Only the expected results are generated automatically by the pseudo-oracle (the functionally equivalent system). When used in partnership with tools for generating test inputs (random or otherwise) it becomes a powerful way to perform high-volume automated testing.

When back-to-back testing is used to support functional testing, the pseudo-oracle does not have to meet the same non-functional constraints as the system under test. For instance, the pseudo-oracle could run far slower than is required for the system under test. It is also not always necessary for the pseudo-oracle to be a complete functionally equivalent system, as back-to-back testing can be performed with a pseudo-oracle that is only equivalent to part of the system under test.

In the context of ML, it is possible to use different frameworks, algorithms and settings to create pseudo-oracles (in some situations it is even possible to create a pseudo-oracle using conventional, non-AI, software). A known problem with the use of pseudo-oracles is that for them to work well they should be completely independent of the software under test. With so much reusable, open source software being used to develop AI-based systems, this independence can be easily compromised.

8.3 A/B testing

A/B testing is a statistical testing approach that allows testers to determine which of two systems performs better[39]. It is often used for digital marketing (e.g. finding the email that gets the best response) in client-facing situations.

As an example, A/B testing is often used to optimize user interface design. For instance, the user interface designer hypothesises that by changing the colour of the 'buy' button from the current red to blue, that sales will increase. A new variant of the interface is created with a blue button and the two interfaces are assigned to different users. The sales rates for the two variants are compared and, given a statistically significant number of uses, it is possible to determine if the hypothesis was correct. If the blue button generated more sales, then the new interface with the blue button would replace the current interface with the red button. This form of A/B testing requires a statistically significant number of uses and can be time-consuming, although tools (often using AI) can be used to support it.

A/B testing is not a test case generation technique as test inputs are not generated. A/B testing is a means of solving the test oracle problem by using the existing system as a partial oracle. By comparing

the new system with the current system, it is possible to determine if the new system is better in some way. When used for digital marketing, the measure of success may be more sales, but for an AI-based system, such as a classifier, the performance metrics, such as accuracy, sensitivity and recall, could be used (see A.8).

A/B testing can be used whenever a component of an AI-based system is updated, as long as acceptance criteria (e.g. 'specified performance metrics must improve or stay the same') are defined and agreed. If A/B testing is automated, then it can be used for testing self-learning AI-based systems, given that valid acceptance criteria are set, by comparing the new performance of the system with its previous performance and reverting to the previous version if the self-learning has not improved the system performance.

8.4 Metamorphic testing

Metamorphic testing[40][41] is an approach to generating test cases that deals, in part, with the test oracle problem often found with AI-based systems, where it is difficult to determine if a test has passed or failed (e.g. because of complexity, non-determinism and probabilistic systems). The main difference between test cases generated using metamorphic testing and conventional test case design techniques is that the expected results in metamorphic testing may not be a fixed value, but, instead, are defined by a relationship with another expected result.

Metamorphic testing uses metamorphic relations to generate follow-up test cases from a source test case that is known to be correct. A metamorphic relation for the software under test describes how a change in the test inputs from the source test case to the follow-up test case affects a change (or not) in the expected outputs from the source test case to the follow-up test case. These metamorphic relationships that are expected to hold can be thought of as partial oracles for the tests conducted.

EXAMPLE 1 A test item measures the distance between a start and end point. The source test case has test inputs A (start point) and B (end point) and an expected result C (distance) from running the test case. The metamorphic relation states that if the start and end points are swapped, then the expected result remains unchanged. Thus, a follow-up test case can be generated with B as the start point, A as the end point and C as the distance.

EXAMPLE 2 A test item predicts the age of death for an individual based on a set of lifestyle parameters. A source test case has various test inputs, including 10 cigarettes smoked per day, and an expected result of age 58 years from running the test case. The metamorphic relation states that if a person smokes more cigarettes, then their expected age of death will probably decrease (and not increase). Thus, a follow-up test case can be generated with the same input set of lifestyle parameters, except with the number of cigarettes smoked increased to 20 per day. The expected result (the predicted age of death) for this follow-up test case can now be set to less than or equal to 58 years.

The expected result for the follow-up test case will not always be an exact value but will often be described as a function of the actual result achieved by executing the source test case (e.g. expected result for follow-up test case is greater than the actual result for source test case).

A single metamorphic relation can often be used to derive multiple follow-up test cases (e.g. a metamorphic relation for a function that translates speech into text can be used to generate multiple follow-up test cases using the same speech input file at different input volume levels but with the same text as the expected result). If metamorphic relations are stated formally (or semi-formally) and source test cases are provided, then it should be possible to automate the generation of follow-up test cases, although it is not possible to automate the generation of the metamorphic relations, which requires some domain knowledge.

The process for performing metamorphic testing is:

a) Construct metamorphic relations (MRs)

   Identify properties of the program under test and represent them as metamorphic relations between test inputs and expected outputs, together with some method to generate a follow–up test case based on a source test case.

b) Review MRs

Review and confirm MRs with customers and/or users.

c) Generate source test cases

Generate a set of source test cases (using any testing technique or random testing).

d) Generate follow-up test cases

Use the metamorphic relations to generate follow–up test cases.

e) Execution of metamorphic test cases

Execute both the source and follow–up test cases, and check that the outputs do not violate the metamorphic relation. Otherwise, the metamorphic test case has failed, indicating a bug.

Metamorphic testing has been used on many types of traditional software, as well as successfully in a wide variety of AI-based application areas, such as bioinformatics, web services, machine learning classifiers, search engines and security. Research shows that only 3 to 6 diverse metamorphic relations can reveal over 90 % of the faults that could be detected using a traditional test oracle[42].

8.5 Exploratory testing

Test design and execution can be conducted in a number of ways, depending on the needs of each project. It can be scripted or exploratory. In practice, a combination of scripted and exploratory testing is typically used, as scripted testing ensures required test coverage levels are achieved and better supports automated testing, while exploratory testing allows for creativity and the rapid execution of tests. When testing AI-based systems, exploratory testing is often found to be beneficial due to its suitability when specifications are poor or lean (such as in agile development).

In exploratory testing, tests are designed and executed on the fly, as the tester interacts with and learns about the test item. Session sheets are often used to structure exploratory testing sessions (e.g. by setting a focus and time limits on each test session). These same session sheets are also used to capture information about what was tested and any anomalous behaviour observed. Exploratory tests are often not wholly unscripted, as high-level test scenarios (sometimes called "test ideas") are often documented in the session sheets to provide a focus for the exploratory testing session.

9 White-box testing of neural networks

9.1 Structure of a neural network

A neural network is a computational model inspired by the neural network in a human brain. It comprises a number of layers of connected nodes or neurons, as shown in Figure 3. Note that in this clause we will use as our example a feedforward neural network, which was the first and is the simplest type of artificial neural network – the only extra complexity we will add is that we will consider a network with multiple layers – known as a multi-layer perceptron (or deep neural net as it has hidden layers).

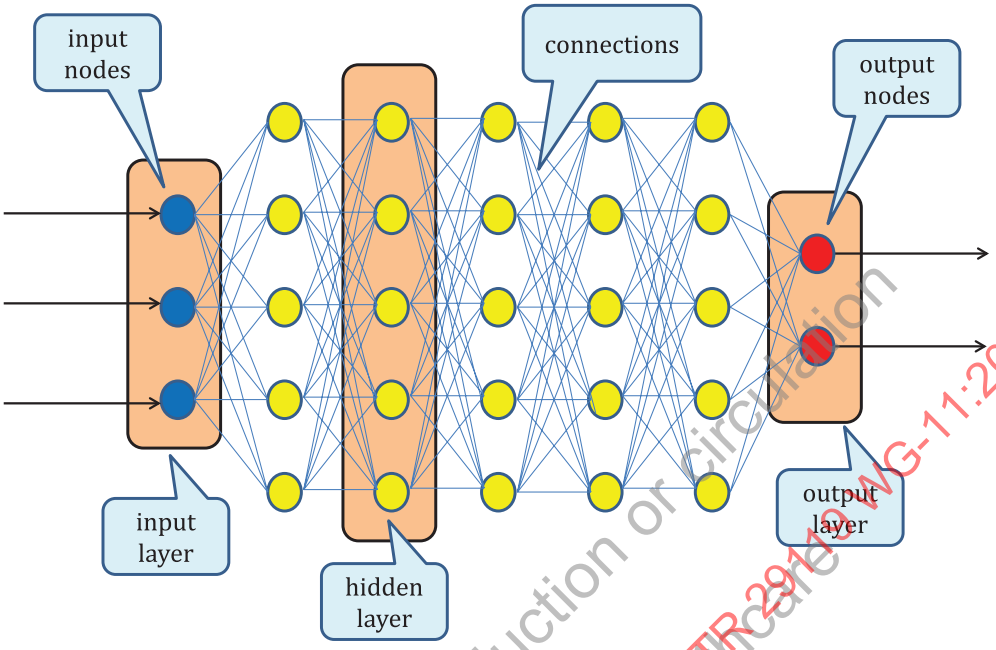

Figure 3 — Deep neural net

The input nodes receive information from the outside world (e.g. each input may be a value for a pixel in an image), and the output nodes provide information to the outside world (e.g. a classification). The nodes in the hidden layers have no connections to the outside world and perform the computations that pass information from the input nodes to the output nodes.

As shown in Figure 4, each neuron accepts input values and generates output values, known as activation values (or output vectors), which can be positive, negative or zero (with a value of zero, a neuron has no influence on downstream neurons). Each connection has a weight (these change as the network is trained) and each neuron has a bias (note that the bias here is quite different from the bias associated with unfairness described in Clause 5.1.5). The activation values are calculated by a formula (known as the activation function) based on the input activation values, the weights of the input connections and the bias of the neuron.

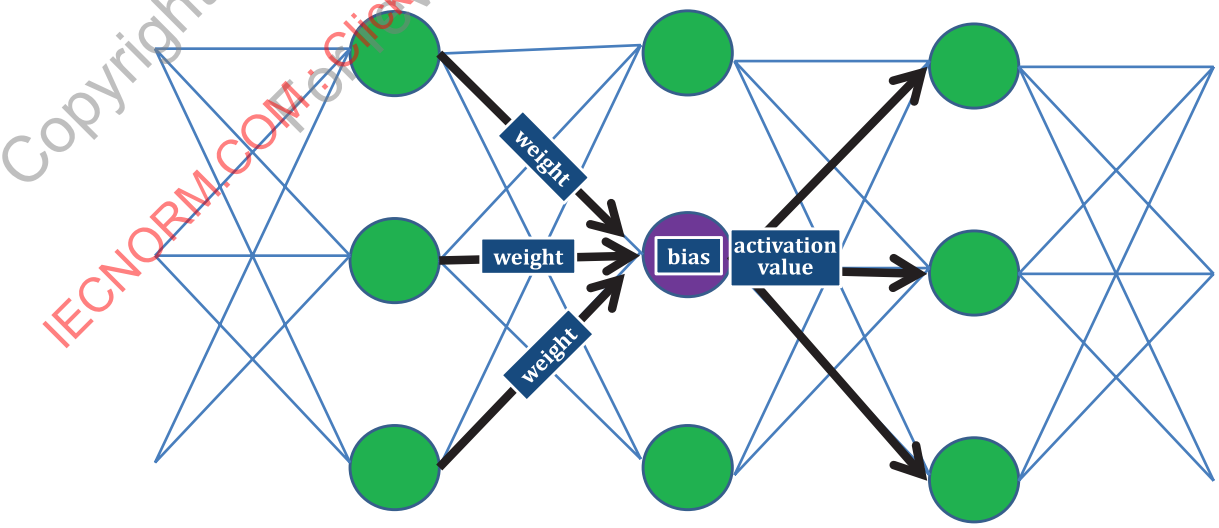

Figure 4 — Neuron activation values

For supervised learning, the network learns by use of backward propagation. Initially all nodes are set to an initial value and the first input training data is passed into and through the network. The

output is compared with the known correct answer and the difference between the calculated output and the correct answer (the error) is fed back to the previous layer of the network and is used to modify the weights. This backward error propagation goes back through the whole network and each of the connection weights is updated as appropriate. As more training data is fed into the network it gradually learns from the errors until it is considered ready for evaluation with the validation data, which will determine the performance of the trained network.

9.2 Test coverage measures for neural networks

9.2.1 Introduction to test coverage levels

Traditional coverage measures are not really useful for neural networks as 100 % statement coverage is typically achieved with a single test case. The defects are normally hidden in the neural network itself. Thus, different coverage measures have been proposed based on the activation values of the neurons (or pairs of neurons) in the neural network when the neural network is tested.

Having measures of coverage of the neural network allows testers to maximize coverage, which has been shown to identify incorrect behaviours in AI-based systems, such as self-driving car systems[43][44].

9.2.2 Neuron coverage

Neuron coverage for a set of tests is defined as the proportion of activated neurons divided by the total number of neurons in the neural network (normally expressed as a percentage). For neuron coverage, a neuron is considered to be activated if its activation value exceeds zero.

9.2.3 Threshold coverage

Threshold coverage for a set of tests is defined as the proportion of neurons exceeding a threshold activation value divided by the total number of neurons in the neural network (normally expressed as a percentage). For threshold coverage, a threshold activation value between 0 and 1 is chosen as the threshold value. Note that this threshold coverage corresponds to 'neuron coverage' in the DeepXplore tool[44].

9.2.4 Sign change coverage

Sign change coverage for a set of tests is defined as the proportion of neurons activated with both positive and negative activation values divided by the total number of neurons in the neural network (normally expressed as a percentage). An activation value of zero is considered to be a negative activation value[45].

9.2.5 Value change coverage

Value change coverage for a set of tests is defined as the proportion of neurons activated where their activation values differ by more than a change amount divided by the total number of neurons in the neural network (normally expressed as a percentage). For value change coverage, a value between 0 and 1 should be chosen as the change amount[45].

9.2.6 Sign-sign coverage

Sign-Sign coverage for a set of tests is achieved if each neuron by changing sign (see 9.2.4) can be shown to individually cause one neuron in the next layer to change sign while all other neurons in the next layer stay the same (i.e. they do not change sign). In concept, this level of neuron coverage is similar to modified condition/decision coverage (MC/DC)[45].

9.2.7 Layer coverage

Coverage measures can also be defined based on whole layers of the neural network and how the activation values for the set of neurons in a whole layer change (e.g. absolutely or relative to each other). Further research is needed in this area.

9.3 Test effectiveness of the white-box measures

There is currently little data on the test effectiveness of the different white-box coverage measures for the white-box testing of neural networks. However, it is generally true that criteria requiring more tests will find more defects than those that require fewer tests, so allowing the relative effectiveness of the measures to be deduced. Several subsumes relationships can be derived from the coverage measures described in 9.2.1. to 9.2.5. All other measures subsume neuron coverage and sign-sign coverage also subsumes sign change coverage. The full subsumes hierarchy for these is shown in Figure 5. Where an arrow points from one measure to another, it means that if the first measure is fully achieved, then the second measure is automatically achieved. For instance, it shows that if threshold coverage is achieved, then neuron coverage is automatically achieved.

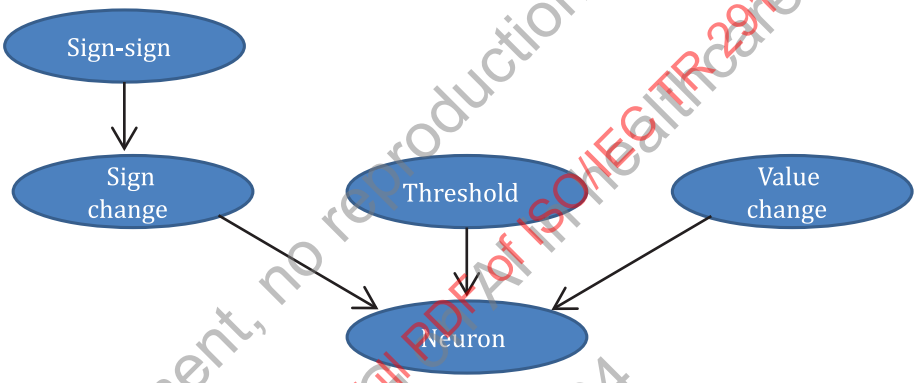

Figure 5 — White-box neural network subsumes hierarchy

Although easy to understand, achieving high levels of neuron coverage can normally be achieved using only a few test cases, so limiting its test effectiveness. Early results for threshold coverage appear to show that this may be a useful measure for generating tests that cover defect-inducing corner cases, but the threshold value may need to be set individually for each neural network. For value change coverage, higher values for the change amount will naturally require more test cases. Sign-sign coverage is normally the most rigorous of the coverage criteria specified here[45].

9.4 White-box testing tools for neural networks

Commercial tools are not currently available to support the white-box testing of neural networks, however there are several research tools, including:

— DeepXplore – specifically for testing deep neural nets, proposes a white-box differential testing (back-to-back) algorithm to systematically generate adversarial examples that cover all neurons in the network (threshold coverage)[44].

— DeepTest – systematic testing tool for automatically detecting erroneous behaviours of cars driven by deep neural nets[46], which supports the sign-sign coverage for DNNs.

— DeepCover - provides all the levels of coverage defined in this clause[45].

10 Test environments for AI-based systems

10.1 Test environments for AI-based systems

The test environments for AI-based systems have much in common with those for conventional systems: typically, the development environment at unit level and a production-like test environment at system and acceptance levels. ML models, when tested in isolation, are typically tested within their development framework, as described in A.2.9.

There are two main factors that affect the selection of test environments for AI-based systems from those required for conventional systems. First, the context in which AI-based systems, such as autonomous systems, operate means their environment can be large, complex and constantly changing. This can make testing in the real world extremely expensive if the full range of possible environments are to be tested, the test environments are expected to be realistic and the testing is to be performed within a sensible timescale. Second, those AI-based systems that can physically interact with humans have a safety component, which can make testing in the real world dangerous. Both factors indicate the need for the use of virtual test environments.

Virtual test environments provide the following benefits, among others:

— The use of a virtual environment ensures that dangerous scenarios can be tested in safety without causing damage to the system under test or any objects in its environment, such as vehicles, buildings, animals and humans. Tests in virtual environments are typically also better for the real-world environment.

— Virtual environments do not need to run in real-time – they can be run much faster with suitable processing power - meaning that many tests can be run in a short time period, potentially decreasing time-to-market by a large amount. A single system can also be tested on many virtual test environments running in parallel, perhaps in the cloud.

— Virtual environments can be cheaper to set up and run than their real-world counterparts. For instance, testing mobile phone communications across widely different urban environments is far cheaper when performed in a laboratory with virtual phones, transmitters and landscapes rather than with real phones being driven around a mix of locations, largely because only the relevant features need to be included in the virtual test environment[47]. However, it should be noted that some virtual test environments must be truly representative and closely match the real-world in some respects. For instance, the testing of pedestrian avoidance in autonomous vehicles can require high levels of image representativeness.

— Sometimes, creating unusual (edge-case) scenarios can be very difficult in the real world and virtual environments allow the creation of such scenarios (and the ability to run multiple variants of these unusual scenarios many times). Virtual environments provide the tester with a greater level of control than they would have with real-word testing. These tests can also incorporate a level of randomness, such as by including AI-based humans in autonomous car testing.

— By supporting the simulation of hardware, virtual environments allow systems to be tested with hardware components even when these components are not physically available (perhaps they have not been built yet) and they allow different hardware solutions to be trialled and compared inexpensively.

— Virtual environments provide excellent observability, so that all aspects of the system under test's response to a scenario can be measured and, where necessary, subsequently analysed.

— Virtual environments can be used to test systems that cannot be tested in their real operational environment, such as a robot working on the site of a nuclear accident or a system to be used for space exploration.

Virtual testing can be performed on simulators built specifically for a given system, but reusable simulators for specific domains are available both commercially and open source, for instance:

— Morse, the Modular Robots Open Simulation Engine, a simulator for generic mobile robot simulation (single or multi robots), based on the Blender game engine[48];

— AI Habitat, a simulation platform created by Facebook AI, designed to train embodied agents (such as virtual robots) in photo-realistic 3D environments[49];

— DRIVE Constellation, an open and scalable platform for self-driving cars from NVIDIA based on a cloud-based platform, capable of generating billions of miles of autonomous vehicle testing[50].

10.2 Test scenario derivation

For the systematic testing of an AI-based system, test scenarios need to be generated to test individual AI components, the interaction of these components with the rest of the system, the complete system of interacting components, and the system interacting with its environment.

Test scenarios can be derived from several sources:

— System requirements

— User issues

— Automatically reported issues (e.g. for autonomous systems)

— Accident reports (e.g. for physical systems)

— Insurance data (e.g. for insured systems, such autonomous cars)

— Regulatory body data (e.g. collected through legislation)

— Testing at various levels (e.g. test failures or anomalies on the test track or on real roads could generate interesting test scenarios for an autonomous car at other test levels, while a sample of test scenarios run on the virtual test environment should also be run on real roads to validate representativeness of the virtual test environment)

An option using combinatorial testing for the generation of test scenarios for the system testing of autonomous cars is described in 8.1. Metamorphic testing (see 8.4) and fuzz testing could also be used to generate test scenarios.

10.3 Regulatory test scenarios and test environments

In the case of safety-related AI-based systems, some level of regulation can apply to the systems. Two options are generally available to government for this regulation; it can allow the development organization to self-regulate or a regulatory body is set up to provide independent assurance that the systems meet minimum standards (a certification approach).

If a certification approach is followed, then the testing approach will need to be shared between the regulatory body and those providing the systems for certification (as it is for the crash testing of cars). A core part of the approach will be shared test environment definitions and shared test scenarios that can be run using test automation on those environments. The core set of shared test scenarios will need to be parameterized to allow new scenarios to be generated by varying the parameter values for each test to prevent overfitting and the regulatory body will also keep a set of private test scenarios that are not shared. The parameterization and the private scenarios should ensure that systems are not built just to pass known tests, and this approach also allows the regulatory body to add new scenarios as they become aware of potential problem situations from actual use of the systems.

Annex A

Machine learning

A.1 Introduction to machine learning

Machine learning (ML) is a form of AI, where the AI-based system learns its behaviour from provided training data, rather than being explicitly programmed. The outcome of ML is known as a model, which is created by the AI development framework using a selected algorithm and the training data; this model reflects the learnt relationships between inputs and outputs. Often the created model, once initially trained, does not change in use. In contrast, in some situations, the created model can continue to learn from operational use (i.e. it is self-learning). Example uses of ML include image classification, playing games (e.g. Go), speech recognition, security systems (malware detection), aircraft collision avoidance systems and autonomous cars.

There are three basic approaches to machine learning (ML), as shown in Figure A.1.

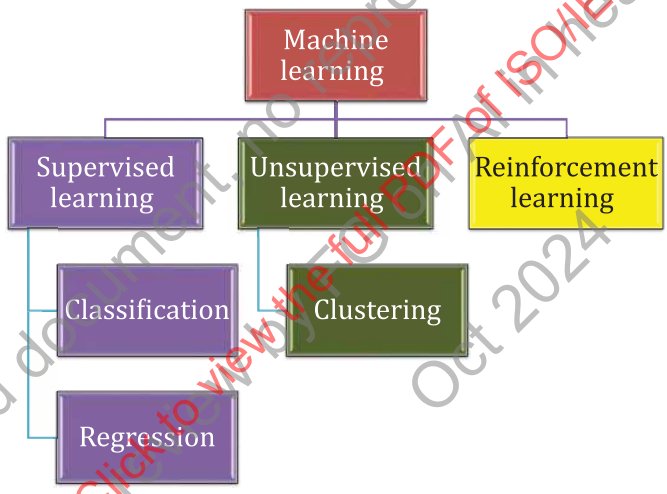

Figure A.1 — Forms of machine learning

With supervised ML the algorithm creates the model based on a training set of labelled data. An example of supervised ML would be where the provided data were labelled pictures of cats and dogs and the created model is expected to correctly identify cats and dogs it sees in the future. Supervised learning solves two forms of problem – classification problems and regression problems. Classification is where the model classifies the inputs into different classes, such as 'yes – this module is error-prone' and 'no – this module is not error-prone'. Regression is where the model provides a value, such as 'the expected number of bugs in the module is 12'. As ML is probabilistic, we can also measure the likelihood of these classifications and regressions being correct (see A.8 on performance metrics for ML).

With unsupervised ML the data in the training set is not labelled and so the algorithm derives the patterns in the data itself. An example of unsupervised ML would be where the provided data was about customers and the system was used to find specific groupings of customers, which may be marketed to in a specific manner. Because the training data does not have to be labelled, it is easier (and cheaper) to source than the training data for supervised ML.

With reinforcement learning a reward function is defined for the system (agent), which returns a higher reward when the system gets closer to the required behaviour. Using feedback from the reward

function, the system learns to improve its behaviour. An example of reinforcement learning would be a route planning system that used a reward function to find the shortest route.

ISO/IEC 23053[51] describes a framework for AI-based systems using machine learning and covers some of the material in this annex in more detail.

A.2 The machine learning workflow

A.2.1 Machine learning workflow overview

The activities in the machine learning workflow are shown in Figure A.2.

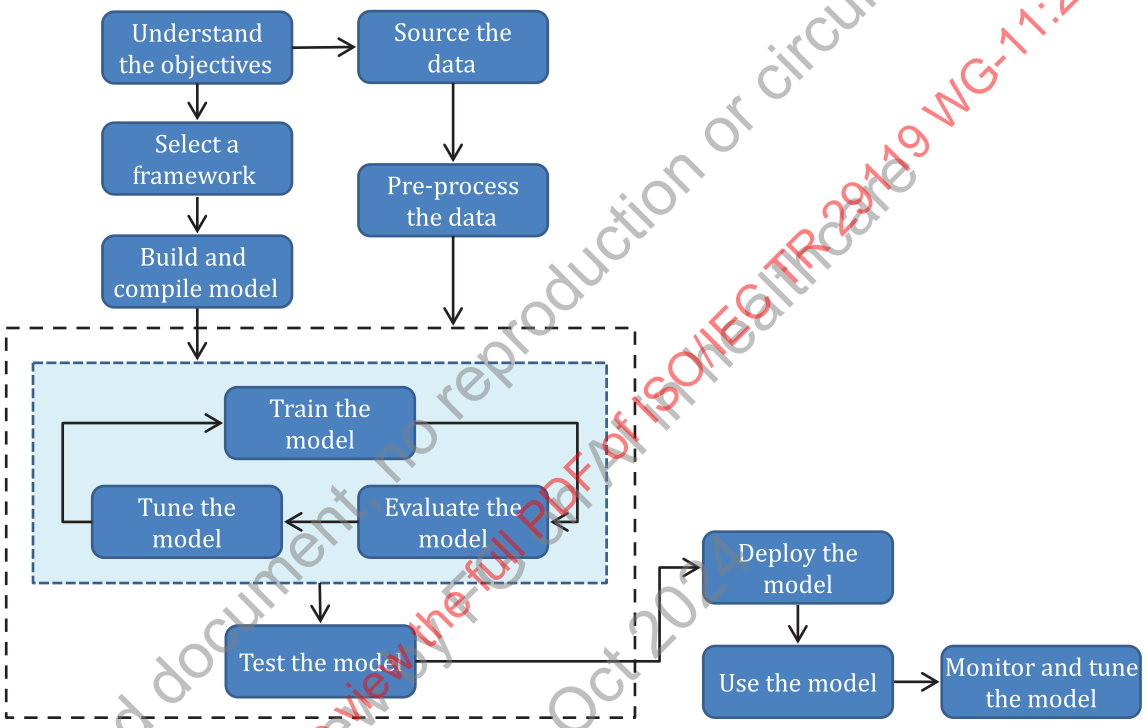

Figure A.2 — Machine learning workflow

The activities in the machine learning workflow are described in A.2.2 to A.2.13.

A.2.2 Understand the objectives

The purpose of the ML model to be deployed needs to be understood and agreed with the stakeholders to ensure alignment with business priorities. Acceptance criteria (including performance metrics – see 4.1) should be defined for the developed model.

A.2.3 Select a framework

A suitable AI development framework should be selected based on the objectives, acceptance criteria and business priorities. These frameworks are introduced in 4.2.6.

A.2.4 Build and compile the model

The model structure (e.g. number of layers) should be defined (it will typically be in source code, such as Python). Next, the model is compiled, ready to be trained.

A.2.5 Source the data

The data used by the model will be based on the objectives. For instance, if the system is a real-time trading system, the data will come from the trading market. If the system is analysing customers' retail preferences for a marketing campaign, then the organization's customer big data will be the source.

The data used to train, tune and test the model should be representative of the operational data expected to be used by the model. In some cases, it is possible to use pre-gathered datasets for the initial training of the model (e.g. see Kaggle datasets at https://www.kaggle.com/datasets). However, raw data normally needs some pre-processing.

A.2.6 Pre-process the data

The features in the data that will be used by our model need to be selected - these are the attributes or properties in the data that we believe are most likely to affect the outcome of the prediction. Training data may need to be managed to remove features that are not expected (or we don't want) to have any effect on the resultant model – this is called feature engineering or feature selection. By removing irrelevant information (noise), feature engineering can reduce overall training times, prevent overfitting (see A.4.1), increase accuracy and make models more generalizable.

Real world data is likely to include outlier values, be in a variety of formats, be missing coverage of important areas, etc. Thus, pre-processing is normally required before it can be used to train (and test) the model. Pre-processing includes conversion of data to numeric values, normalizing numeric data to a common scale, detection and removal of outliers and noisy data, reducing data duplication and the addition of missing data.

A.2.7 Train the model

A ML algorithm (e.g. see machine learning techniques in 4.2.4) uses the training data to create and train the model. The algorithm should be selected based on the objectives, acceptance criteria and the available data.

Note that the activities of training, evaluation and tuning are shown explicitly in Figure A.2 as being iterative, however ML is a highly iterative workflow and it may be necessary to return to any of the earlier activities, such as sourcing and pre-processing the data as a result of later activities (e.g. evaluating the model).

A.2.8 Evaluate the model

The trained model is evaluated against the agreed performance metrics using validation data; the results are then used to improve (tune) the model. Visualization of the results of the evaluation is normally required and different ML frameworks support different visualization options.

In practice several models are typically created and trained, and the best one chosen based on the results of the evaluation and tuning.

A.2.9 Tune the model

The results from evaluating the model against the agreed performance metrics are used to adjust its settings to fit the data and so improve performance. The model may be tuned by hyperparameter tuning, where the training activity is modified (e.g. by changing the number of training steps or by changing the amount of data used for training), or attributes of the model are set (e.g. the number of neurons in a neural network or the depth of a decision tree).

A.2.10 Test the model

Once a model has been trained, evaluated, tuned and selected it should be tested against the test dataset to ensure that the agreed performance criteria are met. This test data should be completely independent of the training and validation data used up until this point in the workflow.