ETSI TR 103 910 V0.0.6 (2023-06)

MTS AI Testing Test Methodology and Test Specification for AI-enabled Systems

<

**TECHNICAL REPORT**

Reference

DTR/MTS-103910

Keywords

AI, ML, testing

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B

Association à but non lucratif enregistrée à la

Sous-préfecture de Grasse (06) N° w061004871

***Important notice***

The present document can be downloaded from:  
<http://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:  
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

If you find a security vulnerability in the present document, please report it through our

Coordinated Vulnerability Disclosure Program:

<https://www.etsi.org/standards/coordinated-vulnerability-disclosure>

***Notice of disclaimer & limitation of liability***

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or

other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.  
The content of the PDF version shall not be modified without the written authorization of ETSI.  
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI yyyy.

All rights reserved.

Contents

Intellectual Property Rights 5

Foreword 5

Modal verbs terminology 5

Executive summary 5

Introduction 5

1 Scope 6

2 References 6

2.1 Normative references 6

2.2 Informative references 6

3 Definition of terms, symbols and abbreviations 6

3.1 Terms 6

3.2 Symbols 7

3.3 Abbreviations 7

4 General conditions of testing ML-based systems 7

4.1 Machine Learning 7

4.2 ML-based systems and its integration 9

4.3 Testing ML-based systems 9

5 Challenges and specifics of testing ML-based systems 10

5.1 Open context and technology 10

5.2 Stochastic solution approach and deep learning 10

5.3 Robustness issue and missing transparency of neural networks 11

5.4 Fault and failure model for testing ML-based systems 11

5.5 Verification vs. validation of ML-based systems 13

6 Quality attributes addressed by testing ML-based systems (Taras) 13

6.1 Model relevance 14

6.2 Correctness 14

6.3 Robustness 14

6.4 Efficiency 16

6.5 Security 16

6.5.1 Confidentiality 16

6.5.2 Integrity 16

6.5.3 Availability 16

6.6 Data Privacy 16

6.7 Fairness 17

6.8 Interpretability 18

6.9 Learnability 18

7 Workflow integration, test methods and definition of test items 19

7.1 A workflow perspective for developing and operating ML-based systems 19

7.2 Overview on test methods for testing ML-based systems 20

7.3 Considerations in defining adequate test items for testing ML-based systems 21

8 Detailed test item identification and definition of test activities within the workflow perspective 23

8.1 Test items of the business understanding and inception phase 24

8.2 Test items of experimentation and training pipeline development phase 25

8.3 Test items of the training phase 27

8.4 Test items of the system development and integration 28

8.5 The test items of the operation and monitoring phase 29

9 Detailed test methods for testing ML-based systems (Uni Göttingen) 29

9.1 Requirements-based testing (Gerhard) 30

9.2 Evaluation of the data (UNI Göttingen) 30

9.3 Risk-based testing (FhG) 30

9.5 Analytical Estimation (HWW) 31

9.6 Search-based testing (Großmann) 32

9.7 Combinatorial testing (Jürgen) 33

9.8 Metamorphic testing (UNI Göttingen) 33

9.9 Differential testing (Universität Göttingen) 34

9.10 Adversarial Attacks (Universität Göttingen) 34

9.11 Exploratory testing (Jürgen) 35

9.12 Probabilistic testing 35

9.13 Testing with failure models 35

9.14 Diversifying test 35

9.15 Reviews 35

9.16 Static analysis 35

10 Workflow and process aspects of testing ML-based systems 36

10.1 Test Management for testing ML-based systems 36

10.2 Dynamic test process for testing ML-based systems 36

10.2.1 Test planning phase 36

10.2.2 Test design & analysis phase 38

10.2.3 Test Implementation & execution phase 40

10.2.4 Evaluating exit criteria and reporting phase 40

11 Testing in the ML Life Cycle 42

11.1 The ML Life Cycle 42

11.2 Plan 42

11.3 Data Engineering 42

11.4 Code 42

11.5 Model Engineering 42

11.6 Model V&V 43

11.7 Test 43

11.8 Integration 43

11.9 Deploy 43

11.10 Operate 44

11.11 Monitor 44

Annex A: Title of annex 45

Annex: Bibliography 46

Annex: Change History 49

History 50

# Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<https://ipr.etsi.org>).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

# Foreword

This Technical Report (TR) has been produced by {ETSI Technical Committee|ETSI Project|<other>} <long techbody> (<short techbody>).

# Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

# Executive summary

The document covers testing of AI-enabled systems for the purpose of standardisation and elaborates on test methodologies and methods for test specification.  
It identifies requirements for testing and comes forward with proposals to tackle the technical aspects of certifying trustworthiness of AI in standardisation contexts.

# Introduction

Machine Learning (ML) and especially the application of neural networks (NN) has been able to achieve amazing successes in recent years due to the availability of large amounts of data as well as the increase in computing capacity. These successes include applications from image recognition, which now achieve better results than humans in many areas, the almost human-like abilities of speech recognition and conversation, which were finally demonstrated convincingly by the NLP model GPT3, or the massive superiority of algorithmic decision systems in learning and playing strategic games such as Go, demonstrated by the Google subsidiary DeepMind.

With the increasing success of ML and NNs, the need to integrate ML models and NNs into software systems that are developed to accomplish critical tasks and operate in critical environments is growing. At this point at the latest, the question arises as to how ML, NN as well as their integration into systems can be rigorously tested and quality assured. This document describes methods and approaches for testing ML-based applications.

We intentionally focus on ML as the currently most widely spread method in the field of artificial intelligence (AI). Other methods, such as Symbolic AI, have their justification, but are not used to the same extent as is currently the case with ML.

The document provides an introduction into the topic of testing ML-based systems. It presents principles and challenges for testing ML-based systems, quality attributes and test itemives as well as suitable test methods and their integration into the life cycle of typical ML-based applications for industry.

# 1 Scope

The present document …

# 2 References

## 2.1 Normative references

Normative references are not applicable in the present document.

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non‑specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.] "ISO/IEC/IEEE International Standard - Systems and software engineering--Vocabulary," in ISO/IEC/IEEE 24765:2017(E) , vol., no., pp.1-541, 28 Aug. 2017, doi: 10.1109/IEEESTD.2017.8016712.

[i.] ISO/IEC 22989:2022 Information technology – Artificial intelligence – Artificial intelligence concepts and terminology

[i.] ISO/IEC TR 29119-11:2020 Software and systems engineering — Software testing — Part 11: Guidelines on the testing of AI-based systems

 etc.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the following terms apply:

**Decision-making process –** A process, that selects a course of action among several possible alternative options. A decision is based on assumptions of the target environment and a set of data that represent a concrete state of the target environment, and a goal to be achieved.

**Deep Neural Network –**

**ML-model –** Software artifact, that has been trained to fulfil a certain task or functionality. During training it processes a set of inputs to learn expectations on its output. ML-models are used for different tasks. In general terms, they are used to support decision-making processes based on input data and a previously learned state. Typical tasks are regression, classification, clustering, dimensionality reduction and control tasks (Zhang et. al. 2019). They are statistic in nature, i.e., solutions based on them are based on statistical inference.

**Neural Network (NN)** **–**Define an ML approach that uses a layered network of mathematically modelled neurons. If an NN has more than one internal layer (so called hidden layer), it is referred to as a Deep Neural Network (DNN).

**Test data sets** are used after training to test the generalizability of the ML model. They are selected independently of the training data but should have the same probability distribution as the training data set.

**Training datasets** are datasets with examples used for learning the patterns and relationships in the data and are used to train the weights of the ML model.

**Training infrastructure –** A software-based infrastructure that enables an efficient training process. It consists of software that supports data selection, data preparation and the compilation of suitable data sets. It also provides algorithms and software to realize different model architectures and operationalizes the training process so that different candidate models can be generated and compared.

**Training process –** A process for building an ML model using a specific training infrastructure and a set of input data or scenarios. It consists of activities that select and prepare the training input in order to tune the model so that it is able to generalizes beyond the training inputs.

**Validation datasets** are used to tune the hyperparameters of a model. In particular, they are used to prevent overfitting of the model to the training data.

## 3.2 Symbols

For the purposes of the present document, the [following] symbols [given in ... and the following] apply:

## 3.3 Abbreviations

For the purposes of the present document, the [following] abbreviations [given in ... and the following] apply:

DNN Deep Neural Network

ML Machine Learning

NN Neural Network

GPU Graphics Processing Unit

MLOps Machine Learning and Operations

DevOps Development and Operations

SVM Support Vector Machines

<ACRONYM1> <Explanation>

# 4 General conditions of testing ML-based systems

## 4.1 Machine Learning

Machine Learning is used as generic term for a sub-field of artificial intelligence, whereby a software system is supposed to find solutions to problems on its own. Based on the information made available to it, such a software system learns to subsequently apply what it has learned to new data. Examples of ML algorithms are neural networks, regression models, decision trees, Bayesian inference and kernel-based methods.

Typically, a differentiation is made between supervised learning, unsupervised learning, and reinforcement learning. Typical areas of application for the latter are real-time decisions, navigation for robots, game playing, and all areas in which the independent acquisition of knowledge and skills is involved [19]. Supervised and unsupervised learning can in turn be divided into two sub-parts, each of which has its own characteristic applications. The two paradigms classification and regression can be assigned to supervised learning. Typical applications for classification are fraud identification, image recognition, customer behaviour analysis and diagnosis. Regression is more typically used for popularity prediction in advertising, weather forecasting, market prediction, lifetime estimation and population growth prediction. Unsupervised learning can again be divided into two sub-paradigms: dimensionality reduction and clustering. Typical applications for the former are big data visualization, compression, structural analysis, feature minimization. Characteristic of clustering are recommendation systems, targeted marketing, segmentation.

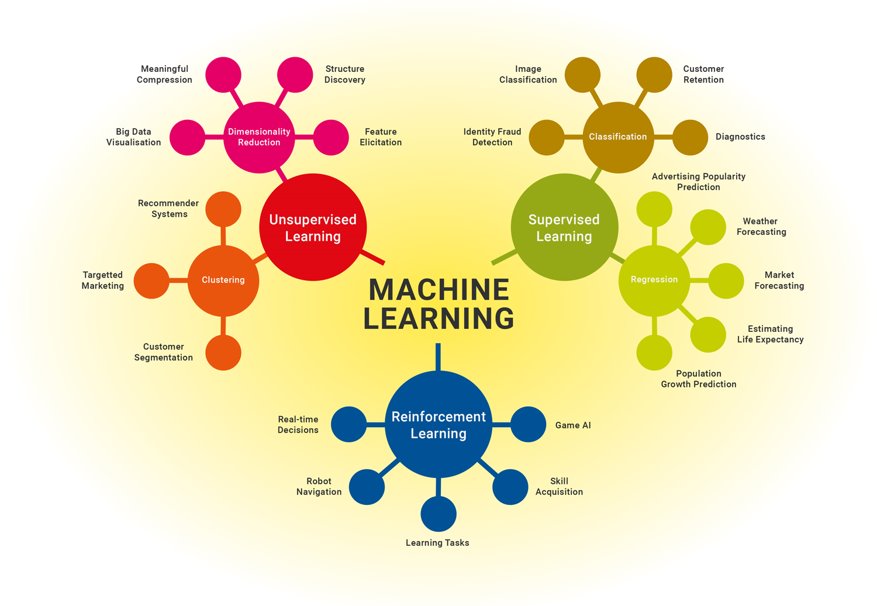


Figure 1 – Different areas in ML and their fields of application

While the functionality of classical software is the result of a design process that addresses the structural set-up of the software, an ML model is built differently. ML is conceptually related to the idea of optimization and to some extent, this has a major impact on testing and quality assurance.

An ML model could be considered as a piece of software with certain structural characteristics. These characteristics, however, describe how parameters are related to each other or algorithms are applied. However, in comparison with classical software, the structural set up of an ML model has only little effect on the actual functionality of the model, probably however on other characteristics like the ability of the model to learn, its robustness, the comprehensibility of the decision-making and other sort of non-functional characteristics.

If we look at NNs, for example, the structural design is quite simple compared to classical software. It consists of a certain arrangement of parameters and algorithms in a graph structure. Parameters and algorithms are arranged in such a way that they are able to approximate the function desired by the user as accurately as possible within the framework of an optimization process based on data. In particular, it is the data, the architecture of the network, the hyperparameters and the way how the training is carried out that are critical to the success of the optimization process. This dependence on data and architecture and the lack of function specific software code has both a major impact on quality assurance in general and testing.

* The software code of an ML model is generic and can be considered quite simple. Thus, it usually does not show the same error probability that classical software has.
* On the other hand, the parameter settings that result from the training process and their interaction during inference are extremely complex and usually not comprehensible to humans. They can be considered as a major origin of failures, but they are nearly impossible to test on a systematic basis.
* The result of an optimization process is to find the most optimal solution possible. For more complex problems, however, these solutions are not error-free. Stochastic deviations and errors are intrinsic properties of ML since it is based on statistical inference.

As a result, a much broader scope has to be set for testing and quality assurance. In addition to the typical white and black box procedure, data and the training process must become the subject of more intensive testing.

## 4.2 ML-based systems and its integration

In the context of quality assurance and testing, we cannot consider ML models in isolation. ML-models are trained, integrated, and applied within a particular technical and often physical environment. Following this, we distinguish the technical environment of an ML model and the application environment. While we usually have influence on the technical environment, the application environment can only be controlled to a limited extent. An ML model in its technical environment can be considered as an ML-based system that has a specific architecture. This architecture implements a typical data processing pipeline. In addition to the ML model, such a system usually contains components for data acquisition and preprocessing as well as components for decision postprocessing and presentation. Since there is an extremely strong binding between the ML model and its environment, the model must especially be tested with the software that is used data acquisition and preprocessing as well as for decision postprocessing and presentation. Unlike classical software, the dependency between the model and its surrounding components is often more difficult to characterize than integration relevant characteristics of classical software.

* ML models are dependent on the input data and their pre-processing. The collection and pre-processing process is done by hard- and software components that thus has a major impact on the performance of the model.
* ML models provide complex output that must be carefully interpreted to lead to a reliable prediction or decision. This is usually done by additional software components that post-process the inference result.
* ML models might be safeguarded and monitored by dedicated software components to ensure a reliable performance over time.
* ML models are trained for a specific purpose, targeting a dedicated operational environment. Deviations between the environment (i.e., the data) used for training process and the operational environment might have crucial effects on the performance of the ML model in operation. Thus, especially the training process must be subject to quality assurance.
* ML models and their properties are often so complex that they are usually not understood.
* Finally, the development of high-quality models requires collaboration from different disciplines. The coordination effort and communication requirements are correspondingly high and must be sufficiently taken into account in the organization of quality assurance.

## 4.3 Testing ML-based systems

Primarily, software testing is an activity that tries to find faults. This can improve the overall quality of the system and reduce the likelihood of undetected failures occurring. Testing, among other things, serves to build confidence in the functionality of a system. In addition to finding errors, this also includes systematic testing, which at least attempts to formulate arguments for the absence of bugs and faults under certain conditions. In analogy to software testing “Machine Learning Testing (ML testing) refers to any activity designed to reveal machine learning bugs.” (Zhang et al., 2019)

On the one hand, this definition shows that testing ML is about quite different and diverse approaches. Testing is not limited to dynamic testing of the model, but also includes testing of the data, hyperparameters and learning algorithms. For this purpose, various methods and approaches can be used, whether they are static like such as review and other forms of analysis, or dynamic in nature. In particular, data is usually not directly testable via a dynamic test and must be quality assured and tested using more suitable analysis procedures.

However, Zhang et al. limit their definition to testing machine learning and do not explicitly address testing ML-based systems. In contrast to that, we want to emphasize that testing ML is always also about testing the software surrounding the ML model. It is therefore not sufficient to ensure that an ML model works as intended as a single component, but always as part of an integrated system.

Thus, *“testing techniques should not solely expose misclassifications and prediction errors at the ML model level, but rather look at the side-effects of such inaccuracies at the overall system level. Individual misclassifications (or individual mis-predictions) are suboptimal definitions of failures if the whole MLS is considered, because they may have no consequences, or, on the contrary, may lead the overall system to deviate significantly from its requirements and result in a failure.”* (Riccio et al., 2019)

In the course of this document, we will work out which test approaches, test itemives and principles can be usefully applied to the testing of ML-based systems. We will investigate which methods of software testing can be directly adopted for ML-based systems and which are difficult to transfer and what needs to be considered additionally. Among many other topics, we will address what role the stochastic nature of ML plays for testing, what and how can be considered a bug in this context, how to deal with specific technical shortcomings of current ML approaches, and which quality properties are relevant, how they propagate through an ML-based system and how they are addressed by different testing approaches.

Testing ML-based systems is the process of planning, preparation, and measurement with the aim of determining the properties of ML-based systems and showing the difference between the actual and the aimed state. (Pol et. al. 2002).

# 5 Challenges and specifics of testing ML-based systems

## 5.1 Open context and technology

ML-based systems are usually used for tasks that cannot be efficiently solved by classical programming. These include problems that are too huge or too complex to be completely specified. This applies, for example, to applications that perform object detection in an uncontrolled environment such as road traffic or the surveillance of a railway line. In this case, the Operational Design Domain (ODD) of such a software is considered as an open context problem. Open context problems are called ¥-complex and cannot be specified correctly in all details (Podey et. al, 2019). Any specification is subject to assumptions that lead to an incomplete or unreliable deduction of the *purpose (i.e. what we may consider as useful service that is to accomplished by a system)*, *context (i.e. the technical and societal environment of a system)* and *realization* of a based system. In addition, state of the art specification processes lack adequate specification means to model this kind of uncertainty in a meaningful way.

This has serious consequences for testing. Given that we cannot fully determine the context of a system, nor, consequently, its purpose, we lack an objective basis for testing the system. Missing specification means to express uncertainty in knowledge during specification puts additional burden on required deductions like deriving test specifications and test implementations that refer to and respect uncertainties in the overall system specification.

Finally model representations of the problem (including the ML-model and thus the ML-based system) are necessarily incomplete, since they are gained by an optimization process that is based on a selected set of examples.

Without exactly knowing the *purpose* and *context* of a problem there is no way to specify completeness with respect to the representativeness of data that are used for training and testing, nor would it be possible to address possible corner cases in a systematic manner.

## 5.2 Stochastic solution approach and deep learning

ML is considered to be a stochastic solution that is often applied to problems, that are intrinsically non-stochastic problems. The recognition of objects, for example, is in principle a deterministic and not a stochastic problem. Stochasticity comes into play because, as already said above, the available knowledge about the purpose and the context of the solution is limited. A stochastic and data-based approach is considered to overcome some of the problems that are associated with the given knowledge gap, but leads to new problems in testing and quality assurance, In particular, the evaluation and treatment of failures must take into account the statistical set-up of the solution approach. Among other things, this includes the fact that failures cannot simply be eliminated and must be accepted within statistical boundaries.

It is assumed that the lack of explicit knowledge about the variety of objects to be recognized can be compensated for by the availability of a sufficient number of examples that implicitly allow this knowledge to be extracted from the examples in the course of a training process. However, this comes at cost. Since no one knows the original distribution of the problem space, examples can only be selected based on a "best guess" about the configuration of the problem space. Moreover, deviations and errors are intrinsic to a stochastic solution approach. Since ML is based on statistical inference, a single failure in a test run cannot be directly counted as an indication to a fault. Thus, it therefore always has to be assumed that a stochastic solution cannot be completely correct in the deterministic sense. There will always be a “natural” error rate that must be accepted. The aim of the optimization process is to reduce this error rate to an acceptable level.

In ordinary test processes, one has a set of test cases and after execution gets the subset of failed ones. To derive a statistical quantity from this, one might look at the relative frequency of failed runs, i.e. the number of failed divided by the number of all test cases.This measure could also be taken in the case of ML systems. But this is not correct, because one expects, due to the statistical nature of SW, that some of the tests will statistically fail. Instead of the relative frequency, one has to weight the individual test cases with their empirical probability, i.e. specify the probability of occurrence to each test case, not only for the failing. Then the total probability for the occurrence of all executed test cases must be calculated and also for the failings. Their quotient gives the correct quality measure for ML system one can derive by dynamical testing.Moreover, approximation methods are only partially reliable, and the generalization capability of any ML solution is limited and susceptible to distribution shifts.

Last but not least, ML models are integrated to form ML-based systems that may consist of a complex interplay between ML-Models and classical software. Considering the tolerances, errors and uncertainties that underlie the processing of data in ML models, the combination of several ML models and their interconnection results in a degree of complexity that far exceeds the complexity of classical software.

ML models cannot be easily fixed or reoptimized at any point, i.e. models may have to be completely rebuilt if deviations occur. Improving the one side can disimprove the other without control.

## 5.3 Robustness issue and missing transparency of neural networks

In contrast to other forms of ML (e.g., linear or logistic regression, the k-nearest neighbour algorithm, Bayesian classifiers, SVM) specifically deep neural networks lack transparency and stability. While interpretable models allow a human user to understand at least parts of the decision-making process, deep neural networks often show a better performance but in the same time the inference procedure lacks interpretability and statistical evaluability. This means that for a human observer, even if he or she has access to the internals of the model, it is not comprehensible on the basis of which parameters and properties in the ML model a particular decision is made.

Furthermore, especially neural networks lack reliable information on the quality of a decision. Although classification or regression models provide prediction probabilities at the end of the pipeline (e.g., by softmax output), these may unfortunately be often misinterpreted as model confidence. However, a model can be uncertain in its predictions even if its softmax output is high [7].  The provision of reliable statements on the uncertainty of a model decision, on the other hand, would make it possible to also design safety-critical applications more reliably. If reliable information on the decision uncertainty is provided in addition to the results, results with high uncertainty could be handled separately by higher-level systems or the user. Moreover, neural networks are not necessarily robust and are vulnerable to intentional and random perturbations. This has been shown in multiple examples through so called Adversarial Examples and the vulnerability of deep learning in the presence of noise. Overall, there seems to be a trade off between robustness and generalizability [15].

## 5.4 Fault and failure model for testing ML-based systems

In classical software testing, a distinction is made between the terms *failure*, *fault* and *error*. While the term failure describes the perceived manifestation of a fault, the term fault describes the internal state of the program that has led to the failure and the term error describes the human cause that led to the fault. The ISTQB distinguishes the terms as follows:

* **Fault (or defect):** a flaw in a component or system that can cause the component or system to fail to perform its required function, e.g., an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system.
* **Failure:** deviation of the component or system from its expected delivery, service, or result.
* **Error:** a human action that produces an incorrect result.

The existence of a failure shows that a system does not work as expected. However, not every fault in a software system shows up by a failure. Faults may have no effect because of the way the software is used, or their effect may be reduced by the shielding or corrective intervention of other software functions so that they do not become apparent. Moreover, failures are not only the result of software errors, but can also be caused by environmental conditions.

Even if the above terms and concepts can be applied to ML, it remains fundamentally necessary to extend them in such a way that the specifics of ML are addressed more strongly.

Zang et al. (Zhang et al. 19) extend the notion of defect to ML by defining that an ML bug (or ML defect) refers to any imperfection in a machine learning item that causes a discordance between the existing and the required conditions. Compared to the definition of a fault, which refers to flaws in components or systems, the definition of Zang et al. extends to so-called ML items, which, in addition to the components and systems, also allow other items from the ML process (eg. data) as carriers of a fault.

The same thing is addressed and extended by Borg et. al. by introducing the terms snug and dug. "Bug is not a suitable term to cover all functional insufficiencies, given its strong connotation to source code defects. Still, we need a new similarly succinct term in the context of MLware. We propose snag to refer to the difference between existing and required behaviours of MLware interwoven of data and source code. The root cause of a snag can be a bug either in the learning code or the infrastructure [36], but it is often related to inadequate training data – we call the latter phenomenon a dug.” (Borg 2020)

Finally, Humbatova et al. (Humbatova et al. 2019) created a taxonomy of ML-faults based on interviews with academics and practitioners in the area of ML. At the high-level, the taxonomy differentiates between ML faults in the various artifacts or work products that are developed during an ML lifecycle. Thus, a distinction is made between faults in the ML model, the API (e.g., to access the GPU or other computation related service routines), the data processing chain (e.g., tensors and input data), and in the different artifacts of the training process (data, hyperparameter).

Failures are usually identified as a deviation between the specification of a system and the actual behaviour of a system. As a prerequisite for such an approach, the specification of a system must be a reliable reference for the expected behaviour. Considering again the application of a system in an open context environment, the specification is not necessarily complete nor completely correct. In the automotive industry, for example, ISO 21448 (SOTIF) is concerned with ensuring the safety of intended functionality (SOTIF) in the absence of a failure. ISO 21448 applies to systems and applications that require adequate situational awareness to be considered safe and the term “absence of failures” is meant to characterize a system to act insufficiently even if it does not get into a specified failure situation. In addition to the absence of failures, such a system is expected to recognize potentially unknown and unsafe conditions and reduce the associated risks by itself. If it is not able to do so, the functionality or behaviour is considered not sufficient for the aimed purpose.

Podey et al. (Podey et al. 2019) distinguish between the so-called

* **aimed purpose** of a system, which is implicitly expected and necessarily vague, and the
* **intended purpose** of a system, relating to explicitly expressed expectations that is for example given by a specification.

Podey et al. use the term **intended** in the same manner than **explicitly expressed** and as applied in the context of ISO26262 & SOTIF in the terms intended functionality and intended behaviour.

Finally, it must be asked whether a stochastic approach, as we find it in machine learning, is not by definition subject to deviations and failures. This can be traced back to two reasons.

On the one hand side, ML is an optimization process that tries to approximate an aimed purpose by adapting a set of parameters to best fit with a given set of data. Separating the data in training, test and validation data sets helps detecting overfitting and allows to measure the generalization capabilities. However, the overall optimization process is a trade-off between different model characteristics and ensures that, on average, a model works as expected. This always implies that situations can be found in which a model decision does not represent an optimum or could even be considered as wrong. On the other hand side the mode of operation of a ML system is statistical reasoning, often according to Bayes. Uncertainties and deviations stem from the statistical nature of the inference process. It is precisely the latter that necessitates the use of statistical criteria to define deviations from expectations.

Thus, a single counterexample may not immediately be considered a violation of the intended purpose. Rather, the failure must be statistically proven as statistical relevant. Whether and which kind of statistical deviations need to be considered as individual failures or not is currently not defined sufficiently.

## 5.5 Verification vs. validation of ML-based systems

The purpose of software verification is to ensure that a software product, service, or system meets a set of design specifications while software validation aims to determine whether such a product, service, or system can accomplish its intended use, goals and objectives [i.1]. Software testing is the process of planning, preparation, and measurement with the aim of determining the properties of a software system and showing the difference between the actual and the required state [4]. In this context, validation testing is considered as an activity that aims to collect evidence that for an end product the stakeholder’s true needs and expectations are met while verification testing checks that all specified requirements at a particular stage of the development of a product are met.

In ML, validation and test have a slightly different meaning. Validation and testing are dedicated activities in the training process of a model. They are often bound to dedicated data sets. **Validation datasets** are used to tune the hyperparameters of a model. In particular, they are used to prevent overfitting of the model to the training data. **Test data sets** are used after training to test the generalizability of the ML model. They are selected independently of the training data but should have the same probability distribution as the training data set. Validation and test data sets belong to the training process and thus are intrinsically bound to the training activities. This is to be distinguished in principle from the analytical activities of testing and quality assurance as normally carried out for software systems. Firstly, the analytical activities are much more far-reaching than just testing the basic performance criteria such as over fitting and generalization. They typically address all the quality attributes that are relevant for a stakeholder. Moreover, they span over a bigger portion of the system life cycle and address all activities that may give rise to quality. Secondly, software validation requires organizational independence in order to achieve trustworthy results. In fact, it is now common practice to have tests performed by somewhat independent departments or teams to prevent bias on the part of the developers.

Nevertheless, there is a common core of both definitions. Evaluation and testing in ML also serves to assess the suitability of a model for its intended purpose. Test and validation datasets serve as more or less ideal representations of the application context and are used test and optimize the parameters of the model to gain the most optimal solution considering evaluation metrics like accuracy, precision, recall, specificity, F1 score, ROC and others.

In summary, there must be a serious shift from verification to validation when testing ML-based systems. Application fields with open context and the stochastic solution approach of ML lead to specifications becoming less informative. For testing, this means that the specification cannot be the only reference for the desired characteristics of a system and new ways must be found to validate the system in a meaningful way. Among other things, the systematic comparison with reference systems or the validation at runtime with a targeted feedback of the user experience as an evaluation criterion could be discussed here.

# 6 Quality attributes addressed by testing ML-based systems (Taras)

AI-based systems can be designed depending on the fulfilment of quality requirements by AI models in tests. For example, improvements in terms of redundancy of functional features, failover or continuous accuracy improvements can be made here. Different learning strategies characterise the methodological spectrum of machine learning, e.g. supervised, unsupervised, semi-supervised, reinforcement or adversarial learning. Compared to purely rule-based methods and within the machine learning spectrum, supervised machine learning enables automatic learning based on data to generate AI models. Below are described the following quality attributes of ML-based systems in detail.

* Model relevance
* Correctness;
* Robustness;
* Efficiency;
* Security;
* Confidentiality;
* Integrity;
* Availability;
* Data privacy;
* Fairness;
* Interpretability;
* Learnability.

## 6.1 Model relevance

In general, different learning strategies characterise the methodological spectrum of machine learning, e.g. supervised, unsupervised, semi-supervised, reinforcement or adversarial learning. Compared to purely rule-based methods and within the machine learning spectrum, supervised machine learning enables automatic learning based on data to generate AI models.



The relevance of models depends on the range of methods and the capabilities of the system used. A basic distinction can be made between three different methods, which are advantageous depending on the concrete application aim. While unsupervised learning is well suited for clustering approaches, reinforcement learning enables the maximisation of a main variable to reach one goal in presence of competing objectives. In terms of supervised learning, machine learning models learn abstract representations of features through a generalisation of function approximation problems. While pure mathematical approximation problems involve real numbers or vectors, interaction spectrum data can be incomplete, imprecise, non-numerical and a mix of different data topologies and structures. The quality of abstraction and applicability to specific problems can be described by the quality of the functional features that underlie the range of existent ML methods and enable a multi-perspective as well as uniform assessment of the range of capabilities.

## 6.2 Correctness

Correctness assesses the extent to which the system produces correct and reliable results. Testing the accuracy of an ML-based system involves comparing its output against the expected or ground truth output.

## 6.3 Robustness

Robustness represents the ability of a model to maintain its performance and accuracy when faced with variations, noise as well as intentional and randomly expected perturbations.

#### Unsupervised learning

**ML method**: In unsupervised learning methods, robustness can be described by false positives, the identifiability of outliers and the ability to create clusters from perturbed data. For this, by cluster stability and outlier detection can be assessed the algorithm’s ability to produce consistent clusters and handle outliers in data.

The quantification of cluster stability can be carried out via metrics like the Jaccard index or Rand index, which enable the identification of similarities between clustering results of data sets including slight perturbations.

Against this backdrop, outlier detection enables to identify anomalous outlier data points. It is essential for applications where detecting rare or abnormal instances is critical. For this, metrics such as precision, recall, or F1 score evaluate the algorithm's performance in correctly classifying outliers.

**Inferred data:** In general, perturbations in the input data can affect the clustering in unsupervised learning. Thereby, noise in the data may lead to erroneous cluster in case of lack of normalisation as well as sensitivity to outliers or noise.

#### Supervised learning

**Training Data**: When assessing the data criteria, the focus can be laid on the sensitivity of certain entity classes in the training data and their impact on the model inference. In the following, effects on class-specific sensitivity of supervised learning models is described by class distribution, feature relevance, data variability as well as quality of labeling.

Metrics such as imbalance ratio, class proportions as well as the number of samples per class can help to forecast potential biases or imbalances that can impact the model's ability to generalise to different classes.

By understanding the feature relevance of the features represented in a dataset, side variables can have a causal effect in the training process to influence the sensitivity of main variables. In this context, after training, while handling unexpected or unusual inputs, a negative impact as such can lead to a loss of accuracy during model inference.

Regarding data variability, high variance as well as types and parameters of probability distributions can affect the sensitivity during inference of a trained model. Thereby, strong correlations between features can make the model sensitive to small changes in the input data and have an impact on the differentiation the individual effects of specific features.

In addition, the accuracy and consistency of labels assigned to the training data can impact the accuracy, bias as well as consistency of results during model inference.

**ML method**: The quality and robustness of supervised learning models is also determined by theorems, procedures, and algorithm-specific metrics employed.

In this context, several characteristics of supervised learning methods emerge as influential factors in assessing model robustness. These characteristics include the extent of regularisation techniques, hyperparameters, model architecture, as well as accuracy metrics:

The robustness can be affected by hyperparameters, which affect the learning process. Thereby, robustness can be influenced by activation functions, batch size, epochs, dropout rate as well as the layer/unit configuration. For instance, by Lasso Regularisation (L1) and Ridge Regularisation (L2), overfitting can be affected during training processes. In this context, by adding weights or shrinking less important features, regularisation is able to affect generalisation.

Regarding the model architecture, robustness can be dependent on the model-dependent structure and design, including layer arrangement, connectivity, activation functions as well as residual connections. Adjustments in architecture, such as layer additions, pruning, neuron counts, or layer types can also have an impact on robustness. Different architectures capture diverse patterns or dependencies in data.

Robustness can also be described by depicting the model's prediction correctness. Thereby, accuracy metrics like Top-k accuracy classification score can reflect the model's ability to handle variations in prediction order.

**Inferred data**: In general, during inference of supervised learning models, perturbation, missing values and outliers can lead to incorrect predictions.

Especially in the case of unfavourable inputs, small changes in data consistency can lead to misclassification or generalisation errors for predictions of multifactorial dependency.

#### Reinforcement Learning

**ML method**: For reinforcement learning, robustness can be depicted via the exploration-exploitation trade-off, reward design, exploration strategies, model architecture, transfer learning, exploration-exploitation noise, and policy evaluation.

By understanding the exploration-exploitation trade-off can be understood, whether a RL model is balancing exploration and exploitation effectively. This is of decisive importance for changing conditions of interaction, while exploration allows the model to discover new states and actions, while exploitation utilises the current knowledge to make optimal decisions.

Moreover, to understand the quality of the reward function, metrics such as average reward can help evaluate the impact of different reward designs on the model's robustness.

Due to different algorithmic characteristics of a model, model convergence can have a significant impact on robustness. Especially with injected noise into the action selection process can be taking an influence on the model’s adaptation to uncertainties in the environment.

**Inferred data:** During inference by reinforcement learning models, noise as well as manipulations can hinder inference procedures. Especially, adversarial attacks on the input data, where malicious agents intentionally manipulate the observations or rewards, can significantly impact the robustness of reinforcement learning models.

## 6.4 Efficiency

Efficiency describes the extent to which resources in terms of, i.a., time, financial expenditures and energy are invested to achieve specific goals regarding, e.g., speed, accuracy, robustness and functional safety. <TBD>

## 6.5 Security

In general, security of machine learning models enables the protection of models and their associated data from various threats and vulnerabilities. In this context, security can ensure a certain extent of robustness of a system against intentional misuse.

#### Unsupervised learning:

**Inferred data:** The quality of inferred data can have a decisive impact on the reliability of the model. With regard to the clustering process, anomalous data points can lead to misclassification during grouing processes. Such data can manipulate the clustering process and lead to biased results. If misleading clusters lead to inaccurate decisions, it can have severe consequences, especially in safety-critical systems, compromising the overall security of the system. For this, the f1 score as well as the area under the receiver operating characteristic curve (AUC-ROC) enable the assessment of the model's ability to correctly identify anomalies and distinguish them from normal data. Additionally, cluster stability metrics, such as the Jaccard index or Rand index, can be used to quantify the consistency of clustering results when the model is applied to different portions of data.

Furthermore, the information security of an unsupervised model can depict its ability to effectively cluster data and handle sensitive information in compliance with information security requirements. For models inferring sensitive or personally identifiable information, privacy evaluation metrics like differential privacy can be used to quantify the level of privacy protection offered by the model.

#### Supervised learning:

**Training data:** As data quality affects the extent to which patterns are learned by an algorithm as well as the reliability during inference, also data biases can be learned by a model, leading to biased predictions. For instance, imbalanced class distributions in the training data can lead to disadvantageous uncertainty, e.g., underfitting or poor generalisation during predictions for underrepresented classes. In this context, adversarial examples as well as anomaleous edge cases can be used tot rain a model proactively to enhance ist security against adversarial attacks.

In terms of information security, existent sensitive or personally identifiable information in the training data can affect the information breaches.

**ML method:** As ML method-specific parameters are able to directly influence the model’s behaviour and performance, regularisation, hyperparameters as well as the model architecture can have a significant impact on the security of a supervised learning model. In this context, to ensure security by focussing on the algorithm behind ML models, a balance between accuracy and reliability is key. Such a balance can be achieved via a composition of the following three factors. First, selecting appropriate regularization techniques is essential to prevent overfitting and improve the model's ability to generalise to new data. Second, tuning hyperparameters ensures efficient and effective model training, optimising the learning process. Lastly, the specific model architecture, which represents the underlying AI method translated into software, plays a vital role in shaping the model's behavior and performance.

**Inferred data:** Similiar to unsupervised learning, the presence of adversarial examples is able to manipulate model's inference. In case of supervised learning models, adversaries can result in malicious as well as biased predictions. Additionally, the processing of sensitive information during inference plays a decisive role in terms for information security.

#### Reinforcement learning:

**ML method:** In general, while manipulating exploration, the model does not discover new states and actions, while exploitation can not be carried out in the most efficient way. Such a manipulation can lead to malicious behavior, making the model vulnerable to adversarial attacks resulting in misleading rewards.

**Inferred data:** During adversarial attacks similar to unsupervised as well as supervised learning models, inferred data can introduce perturbations into the environment to distort sampling processes with a negative impact on decision-making.

### 6.5.1 Confidentiality

Confidentiality of ML-based systems depicts a set of characteristics to ensure the information security of data as well as the ML-based system. In terms of confidentiality assessment access control mechanisms, extent of anonymisability of data involved, compliance with legislations, vulnerability assessments as well as the system security can be evaluated.

### 6.5.2 Integrity

In the context of machine learning, integrity means that the data used in machine learning models meet certain operational quality requirements as well as information security requirements. Operational quality requirements represent data that is complete, consistent and free of bias, which requires data collection with multi-factor dependencies, cleaning and validation processes to avoid errors, inconsistencies or biases in the data.

In addition, integrity in terms of operational quality can be represented by the degree of transparency and explainability so that, among other things, the planning, decision-making, optimisation and inference processes performed by the model can be understood and explained by humans.

In terms of information security requirements, machine learning models and the data they use must be protected from unauthorised access and modification.

### 6.5.3 Availability

In general, the availability of ML-based systems describes their ability to operate with a certain quality of feedback, controllability and systematic embedding of influencing factors. To assess the feedback quality, individual functional characteristics can be tested, such as the absence of interruptions, appropriate time intervals or the degree of implementation of specified parameters. The testing of availability can be carried out within the framework of load tests, fault injection or response time analyses, among others. Load tests can assess the extent to which ML models can respond to an increasing number of parameters, depending on the ML method used and the characteristics of the data analysed. A system's response to parasitic behaviour can be analysed during fault injection to assess how it reacts.

## 6.6 Data Privacy

Privacy describes protection of personal or sensitive information from access or use by unauthorised individuals or organisations. In ML-based models, privacy depicts restricitions to ensure information security during obtaining training data, training the model and analysing data during model operation.

## 6.7 Fairness

The fairness of machine learning models is influenced both by the algorithms used to develop models and by the data if used as the basis for training and testing the models.

#### Unsupervised learning

As a metric to depict deterministic outcomes during the separation of entities in dependence of sensitive characteristics can be used "cluster stability," which measures the degree to which the model produces consistent clustering results when applied to different random subsets of data. This metric can be used to identify the variability in the model's assessment of favours or disadvantage of specific entitites.

Using cluster separation can be understood to which extent different entities or correspondent sensitive characteristics are separated into distinct clusters. By that can be clarified, how sensitive attributes are grouped together or devided into separate clusters by using, i.a., the metrics „Davies-Bouldin index“ or „Silhouette score“.

The extent to which entities are homogeneously represented within each cluster can be determined by focussing on their sensitive characteristics. Metrics such as epsilon-neighbourhood and within-cluster variance may be used for this purpose.

#### Supervised learning

**Data:** With regard to data, consistency, representativeness as well as diversity are key to affect fairness for testing and training of ML models.

Consistency can be measured by assessing the degree of agreement via Cohen’s kappa between different labels or annotations for the same data point. A high level of agreement indicates consistency and reliability of training data.

Representativeness can be measured by assessing the degree to which the training data accurately reflects the distribution of data in the real world. One way to measure representativeness is to use "cross-validation", which involves evaluating the model's performance on a diverse range of subsets of the data. By evaluating the model's performance on a range of data subsets, researchers can ensure that the model is trained on data that is representative of the full range of real-world data. Moreover, by evaluating the model's fairness characteristics on a range of environments, it can be ensured that the model is trained on data that is representative of the full range of real-world scenarios that it is likely to encounter.

To foster diversity, bias can be affected by ensuring that data is unbiased by identifying and removing sensitive characteristics which might foster bias. Due to changes in environmental conditions as well as tendencies in multifactorial development of forecasted parameters, machine learning models may need to be retrained or updated to ensure that they continue to produce results which may lead to fair decisions.

**Algorithms:** By focussing an algorithms of supervised learning, equalised odds might be achieved to avoid unfairly favoring or disadvantageous behaviour towards sensitive characteristics by balancing both true and false positive rates along entities involved.

Furthermore, statistical parity might assess whether the overall probability of a positive outcome is equal across different entities in dependence of their sensitive characteristics by measuring the distribution of positive predictions.

Additionally, analysing the confusion matrix separately for specific entities and correspondent sensitive attributes provides insights into potential biases in supervised learning algorithms. For this purpose can be compared the true positive rate, false positive rate, false negative rate, and true negative rate across entities involved.

#### Reinforcement learning

For reinforcement learning, consistency can be measured by assessing the degree to which the model produces consistent behavior in response to similar input stimuli over multiple episodes. One way is to measure the extent to which the model produces similar rewards for similar actions in different episodes. This metric can be used to evaluate the stability of the model's behavior over time to identify correlating factors which might lead to unfair decisions.

To describe the deviation from intended rewards, the believe-action gap of RL models might be identified to understand, how the assumed intention of the model regarding beliefs is reflected by its actions. The believe-action gap enables to understand disparities in the distribution of actions across sensitive characteristics which might foster bias in decision-making. In addition, comparing the rewards for individual entities, as well as differences that exist between them, enables an assessment of fairness in the allocation of rewards.

Furthermore, to understand fairness characteristics of RL models with correspondend multifactorial relationships, agent variation might be taken into account, to depict long-term consequences regarding disproportionately favor or disadvantage sensitive characteristisc. This involves acknowledging and addressing variations in starting conditions, capabilities, or preferences of the agents within the RL system.

## 6.8 Interpretability

Interpretability of ML-based systems refers to the ability to understand and explain systematics behind the system's predictions or decisions. It is an essential aspect of building trustworthy and reliable AI systems.

## 6.9 Learnability

<TBD>

# 7 Workflow integration, test methods and definition of test items

In industry, there are a lot of established workflows that describe activities for software development and machine learning. In the context of software development, workflows range from the classic waterfall model to agile variants and DevOps. In the area of machine learning and data science, workflows have been established that focus on data preparation and training. For example (Akkiraju 2017) describe a reinterpretation of the Software Capability Maturity Model (CMM) for the machine learning model development process. (Akkiraju 2017) describe a reinterpretation of the Software Capability Maturity Model (CMM) for the machine learning model development process. (Amershi et al. 2019) summarizes the experiences of several Microsoft software development teams into a nine-step workflow for integrating machine learning into application and platform development. Based on CRISP-ML, Studer et al. (Studer et al. 2020) propose CRISP-ML (Q), a process model for the development of machine learning applications extended by quality assurance activities. It defines tasks that span the entire life cycle of an ML application. For each task, a quality assurance methodology is presented that is based on practical experience as well as scientific literature and provides a solid foundation for holistic quality assurance. Combining workflows and ideas from software engineering and machine learning can provide a solid foundation for developing AI-based applications.

## 7.1 A workflow perspective for developing and operating ML-based systems

Ein Bild, das Text, Screenshot, Diagramm, parallel enthält.

Automatisch generierte Beschreibung

Figure 2 – Development and training workflow to develop, train and deploy ML-based systems or applications

In the context of identifying and locating important quality assurance activities, this document introduces a workflow model that encompasses both the perspective of classical software engineering and the data science activities of machine learning. When defining the workflow model, design activities for the overall system and individual components were not mapped. Instead, software development activities that are required for the provision of highly automated training infrastructures are considered. Figure 2 shows the abstract definition of the workflow including classical software development activities, as well as typical data science activities like data preparation, training, and validation. The workflow is based on the activities known from the established workflow models for traditional software development and data science mentioned above. It describes the main activities and artifacts from both domains and as such describes the development, integration and operation of an ML-based application as an integrated software product consisting of ML models and traditional software.

On the high-level the workflow distinguishes four different phases that are differentiated and detailed in Figure 2 and explained below. Each of these phases are defined by a set of activities that are roughly assigned to the field of data science (blue), software development (grey), and integration (orange).

1. **Business understanding and inception** aims to derive a basic understanding of the overall objectives and requirements of the ML-based system. For this purpose, it is necessary to understand the business and technical context of the system and to obtain a basic understanding of the data available for modelling.
2. **Experimentation and training pipeline development** aims to evaluate the data and modelling approach and to build a modelling infrastructure. In this phase, PoC systems are developed and evaluated for their basic applicability. Depending on the modelling approach, the training and data preparation pipeline is developed and integrated.
3. **Training** aims to create new models based on the modelling approach and with the help of the training pipeline. Depending on the degree of automation available, activities for data preparation, training including the tuning of hyperparameters, validation and quality assurance of the model are executed more or less automatically.
4. **System development and integration** aims to integrate the ML model into a software environment. The complexity of the integration depends on the application context and ranges from the simple provision of a user interface to complex integration with other models, sensor systems and complex control software, such as in automated driving.
5. **Operation and monitoring** is finally the phase in which the integrated ML-based system is being executed and **monitored** in its operating environment. Depending on the application context, various operating environments are possible, ranging from a simple cloud deployment to a distributed edge deployment.

Most of the phases end with a dedicated integration activity (depicted in orange), integrating the key work products and as such defines the main artifact that is propagated or deployed to the next phases (green arrow).

## 7.2 Overview on test methods for testing ML-based systems

The work products of a given workflow phase and their systematic integration are usually the subject of systematic testing. Testing is considered here as the process of evaluating a software system or component to determine deviations between expected and actual behaviour. The main objectives of testing are to detect bugs, verify functionality, and ensure that the software meets the specified requirements.

Testing is usually performed during the development phase or as a special quality assurance measure prior to deployment (Phase 1 – 4 in Figure 2), but can also be performed during operation (Phase 5 in Figure 2). The latter becomes necessary especially for systems with strong dynamics or for systems with a high dependency on the environment. Basically, a distinction can be made between dynamic and static testing methods.

1. In **dynamic testing**, the system is executed. Specific inputs or test cases are applied as inputs to the running system and the observed results are compared with the expected results.
2. In **static testing**, the system is not executed or artifacts that cannot be executed are examined. These include specifications, architectures as well as data. Static testing can be done automatically with the help of dedicated **analysis tools** or manually through **review**.
3. **Monitoring** is a testing method that does continuous observation and measurement of a software system during its runtime. It involves the collection and analysis of real-time data about the system's performance, behavior, and various **operational metrics**. Monitoring helps identify potential problems, bottlenecks, or anomalies that may affect the availability, performance, safety, security of the system. It provides insights into system health, usage patterns, resource utilization, and other relevant aspects.

In summary, review, analysis, dynamic testing and monitoring are all considered as useful testing methods to test ML-based systems. Static and dynamic testing often focuses on assessing the correctness, functionality, and compliance of software systems before deployment, while monitoring concentrates on real-time observation, measurement, and analysis of the system's performance during runtime. All these activities are considered crucial for maintaining software quality, reliability, and overall system health for classical software systems as well as for ML-based systems.

## 7.3 Considerations in defining adequate test items for testing ML-based systems

The term test item describes the item to be tested by a particular test method. In the case of dynamic testing, this is normally referred to as System Under Test (SUT), which somehow highlights the dynamic nature of the test item. However, analogous to the ISTQB, we use the concept of a test item in the following, which includes any work product in the life cycle of an ML-based system, in order to clarify that we deal with both static and dynamic test procedures.

Although our primary test item, as the name of this report suggests, is the ML-based system, we obtain several other test items that can be tested individually or partially integrated considering the development of an ML-based System as well as its systematic integration from individual components.

Due to the high importance of the data and/or the training process, the literature explicitly distinguishes between test items of the training phase, which are crucial for the quality and properties of an ML model, and the development and runtime artifacts, which are relevant for the composition and integration of an ML-based system based on individual components. Zhang et al., 2019 for example distinguishes on a high-level between testing data, testing the learning program (i.e., the training infrastructure) and testing the ML-framework (i.e., the libraries and building blocks that are used to define models).

“*Thus, when conducting ML testing, developers may need to try to find bugs in every component including the data, the learning program, and the framework.“* (Zhang et al., 2019).

In Section 8 we, the different test items are systematically derived along the workflow defined in Figure 2. As already mentioned before, test items are normally the work products of a given workflow activity or phase. At this place we will start with a more general overview on considerable relevant test items.

Test items are:

1. **Specifications, requirements and planning documents:** Before a system can be meaningfully constructed or optimized, it is necessary to determine what the system is to accomplish, how it is to be structured, and how the necessary processes are to be planned. Testing these specifications, requirements and planning documents is mainly done by reviews and has to consider the different view points and terminologies in software engineering and data science.
2. **Data:** Unlike in traditional software development, data and its provision as datasets for training, testing and validation are one of the most important artifacts in machine learning. Testing of the data can be realized via different methods. These include reviews, static and statistical analysis, directed data testing by operationalizing the data through test and analysis models. This involves testing the data structure, its markup and metadata, as well as its meaning.
3. **Development, modeling and training infrastructure:** Especially with regard to the automation of particularly complex processes such as data preparation and the tuning of relevant hyperparameters, as well as training, automation and tool support are usually relied on. Nowadays, we speak of pipelines when there is a tool chain that automates more complex processes. Since these infrastructures have a high impact on the quality of an application or a product and usually have to be rebuilt and tuned for new products and applications, the testing of these infrastructures is a necessary requirement.
4. **Models:** Models are the main result of the training phase. The testing of models ensures that a model meets the requirements placed on it. Requirements are usually formulated by KPIs along various quality dimensions. Testing of a model is usually done dynamically by feeding a variety of test data into the model and comparing the actual results with the current results. Errors are usually quantified and qualified using statistical measures.
5. **The ML-based system:** Finally, the resulting software system must be tested across its integration stages. This includes the individual software components, their partial integration, and the integrated system in the various execution environments. In view of the fact that this is an ML-based system, ML components such as the model or the integration of the model with its pre- and post-processing components (prediction pipeline) are mentioned separately. For testing, a variety of test methods are used, i.e. dynamic testing, static analysis, reviews as well as various monitoring activities at runtime.

# 8 Detailed test item identification and definition of test activities within the workflow perspective

Test activities range from testing the individual test items and their integration to larger items in the integration phases. Near all phases of the workflow depicted in Figure 2 end with a dedicated integration phase having work products associated that are subject to dedicated testing activities. However, also intermediate work product are of interest for testing. Figure 3 shows the development and training workflow specified in Figure 2 extended by dedicated testing and monitoring activities.

Ein Bild, das Text, Screenshot, Diagramm, parallel enthält.

Automatisch generierte Beschreibung

Figure 3 – Development and training workflow extended by testing and monitoring activites

Testing activates are denoted in green and monitoring activities are denoted in light blue. Please note, that also typical data science activities like Data Validation, Model Evaluation, and Model Validation include dedicated testing activities. These activities will be discussed in relation to the general testing activities denoted in green, since there are sometimes the same and show a larger amount of overlap in approaches, methods and results.

The rest of the text identifies the key work products and acceptance criteria for each phase of the workflow. Each work product can then be considered as an independent test item to which suitable test methods and objectives are assigned. Finally, each combination of test item, acceptance criteria and test method can then be assigned to the testing acivities in Figure 3.

## 8.1 Test items of the business understanding and inception phase

Ein Bild, das Text, Screenshot, Schrift, Visitenkarte enthält.

Automatisch generierte Beschreibung

Figure 4 - Testing activities in the phase of business understanding and inceoption.

The **Business understanding and inception** phase aims for deriving and integrating the major KPIs and requirements of the application, service or system. Major work products are the business related KPIs, the technical KPIs and the overall requirements and quality criteria. The activity *Planning and Requirements* address general requirements management and planning activities while the activity *KPIs and Requirements Integration* addresses in particular the harmonization of KPIs and requirements with regard to completeness consistency, absence of contradictions and other cross-cutting concerns. Considering the iterative character of ML, KPIs and requirements need to be adapted in the following phases.

*KPIs and Requirements Review* is considered a testing activity that checks individual KPIs and requirements for correctness, realizability, completeness, etc. and sets of KPIs and requirements completeness, consistency and absence of contradictions and other cross-cutting concerns.

Table 1 provides an overview on the major work products of the business understanding and inception phase, the related acceptance criteria and items.

Table 1 Work products, acceptance criteria and test types for the business understanding and inception phase.

|  |  |  |
| --- | --- | --- |
| **Work product/test item** | **Acceptance criteria** | **Test method/test objective** |
| Business KPIs | * Business KPI are correct, complete, consistent, unambiguous, measurable, traceable, feasible and validated. | * Review of business KPIs |
| Training KPIs and acceptance criteria for training | * Training KPIs and acceptance criteria for training are correct, complete, consistent, unambiguous, measurable, traceable, feasible and validated. | * Review of training KPIs |
| Requirements and quality criteria | * Requirements and quality criteria are atomic, correct, complete, consistent, unambiguous, verifiable, traceable and validated. | * Review of data quality criteria |

## 8.2 Test items of experimentation and training pipeline development phase

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Figure 4 - Test activities in the phase of experimentation and training pipeline development.

The experimentation and training pipeline development phase consists of extensive activities in the area of Data Analysis and Model Analysis. The purpose of these activities is to identify suitable modelling approaches and data preparation procedures that can be used to meet the KPIs and requirements derived from the first phase for the given data set. In the course of the activities, a suitable model architecture including layers and model code will be realized and the necessary software components for data preparation and training will be implemented and integrated into a functional pipeline. Major work products of this phase include the adequate data format for training data, samples of the training data, feature definitions and feature selection criteria, the model architecture and code as well as all algorithms, libraries and components required for the training.

*Data Structure Testing and Feature Testing:* Data structure testing is a test activity in which syntactic properties of data and data sets are checked. These include the correctness and properties of data formats and data types, the metadata and its availability, and annotation formats for labels and other data annotations. Feature testing includes testing of feature relevance, compliance, ranges as well as tests for the general availability and costs for certain features.

*ML-Framework Testing:* ML-Framework Testing is considered an activity that tests the functionality, reliability and scalability of the training environment. This includes testing of libraries that provides training algorithms like loss functions and optimizers as well as the code that allows to compose models out of predefined building blocks.

*Model Structure and Unit Testing:* Includes the test of the synthesized model structures and model code. This includes the code of the individual layers including their functions, the integration of the layers and the data flows and data type compatibilities between the layers as well as the integration of the model into the ML framework.

*Training Pipeline Testing:* This testing activity includes testing of all components that are part of the training process. This includes testing the relevant components for data gathering, data preparation, and feature generation/extraction, testing the the ML-Framework and the model structure and code as mentioned above, and testing the monitoring and validation components, that are meant to safeguard the training process in the training phase. Testing covers all integration stages, starting with unit/component testing, through integration of individual components, to testing of the entire pipeline.

*Experiment Monitoring*: Experiment monitoring is used to capture information gained during data and model analysis to ensure systematic decision making and traceability in the transition of POC models and infrastructures towards an efficient production environment.

Table 2 provides an overview on the major work products of the **experimentation and training pipeline development** phase, the related acceptance criteria and testing types.

Table 2 Work products, acceptance criteria and test types of the experimentation and training pipeline development phase

|  |  |  |
| --- | --- | --- |
| **Work product/test item** | **Acceptance criteria** | **Test type/test objective** |
| Training data format and samples. | * Quality criteria for data quality are completely defined. * Training data are suitable for purpose (training and inference) * Training data are available. * Training data are processable | * Review of data quality criteria * Testing initial samples of training data for major data quality attributes * Review of data sources and their availability * Testing training data formats and meta data |
| Features and feature selection criteria | * Features are identified. * Features are sufficient to allow for reliable inference. * Features are available in training and inference data | * Redundancy? * Ranking? / Usefulness? |
| Label structure and label adequacy | * Labels are identified. * Label structure and format is adequate? | * Review label structure and format * Testing label completeness * Testing label adequacy |
| Model architecture, layers and algorithms | * The basic model architecture, layers and algorithms are defined and evaluated with the data that are available for training and inference | * Review of architecture and layer interfaces. |
| Training algorithms (Loss Function, Optimizer), libraries and interfaces | * Algorithm used for training are working correctly. * Test the libraries and interfaces used for training and model set up are compatible with each other and the machine learning model being developed | * Review of algorithms * Code review * Functional testing of algorithms and libraries * Compatibility reviews and tests of training and library interfaces |
| Model Code | * Model code is sufficiently tested with respect to training and inference capabilities and layer integration. | * Code review of model code * Layer and submodel testing (unit testing) * Functional testing of model software behavior during training and inference * Metamorphic / Differential? |
| Hyperparameters | * Major hyperparameters are defined and tuned for the given data and model architecture | * Cross Validation to test the performance of the model on different subsets of the data and with different hyperparameters. |
| Basic model performance | * ML-Model performance is sufficient as a candidate model for exhaustive training. * ML-Model is robust and generalizes well. * The ML-model is free of unwanted bias | * Model performance testing and evaluation * Model robustness testing * Model bias testing |
| Training pipeline components | * Functionality of the pipeline components * Integration of the pipeline components * Software-hardware embedding of the training pipeline | * Unit/component testing of pipeline components (classical software testing). * Integration testing of pipeline components (classical software testing). * System testing of the training pipeline (classical software testing). * Testing of Software-hardware embedding (e.g. GPU integration) of the pipeline. * Test the API of the pipeline to ensure that it is easy to use and integrates well with other systems. |

## 8.3 Test items of the training phase

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Figure 6 - Test activities in the phase of model training

The training phase is responsible for training ML models for production based on the modelling approaches and data preparation activities identified in the experimentation phase. If possible, this is done in an automated way and with the help of a predefined training pipeline. In the pipeline, all necessary activities from data validation and extraction, data preparation, model training, model evaluation, and model validation are performed. The dinal result is the delivery of an ML model that best meets the requirements and KPIs from phase 1.

*Data Testing:* Data testing is an activity to detect errors in the data, the composition of the data sets, and the distributions of properties, features, or other characteristics in the data. Data testing can be very diverse and includes tests with different data compositions, statistical and structural analysis of the data, and monitoring of predefined KPIs for different quality characteristics of data.

*Model Testing*: Model Testing is the activity to identify deviations of the actual model performance from the expected model performance as well as to identify systematic errors in the model. This includes activities like measuring the accuracy and robustness by train/test split, cross validation, and other methods. Often there is an overlap in methods and approaches with the model evaluation and model validation phases. However, the latter are meant to select the best models and architectures from a given set of models, while model testing tries to check of the acceptance criteria for a given model is met.

*Training Monitoring:* Is the activity to collect data during data preparation and training. These data are used to track dependencies (traceability) between data, hyperparameter settings and the resulting models. Moreover, these data can be used to continuously track quality related data and thus serves as a data source for localizing errors and track the state of certain quality attributes (e.g. number of training data failures an deviations etc.)

Table 3 provides an overview on the major work products of the **training** phase, the related acceptance criteria and testing types and test objectives.

Table 1 Work products, acceptance criteria and test types of the training phase

|  |  |  |
| --- | --- | --- |
| **Work product/test item** | **Acceptance criteria** | **Test type/test objective** |
| Training data | * Data and data sets are correct, * Data distribution and data splits are defined correctly. * Data are free of unwanted bias | * Test data format and type correctness * Test data correctness and consistency * Test data sets for missing data, duplicates, outliers, inconsistencies * Test data set distribution and data skewness (e.g., any kind of imbalance regarding features and labels) * Test for correlated features * Test data for unwanted bias |
| Hyperparameters | * Hyperparameters are fine-tuned | * Cross Validation to test the performance of the model on different subsets of the data and with different hyperparameters (e.g., different learning rates, batch sizes, regularizations, etc.). |
| ML-Model | * ML-Model performance is sufficient for production. * ML-Model is robust and generalizes well. * The ML-model is free of unwanted bias | * Model performance testing and evaluation (i.e., evaluating various performance measures such as accuracy, precision, recall, F1-score, AUC-ROC, mean average precision, or any other relevant metrics specific to the problem domain) * Model robustness testing * Bias and Fairness Assessment |
| Evaluation concepts and criteria | * The evaluation concept and criteria are sufficient to ensure an adequate selection and evaluation of the candidate models. | * Review of evaluation concept and criteria |

## 8.4 Test items of the system development and integration

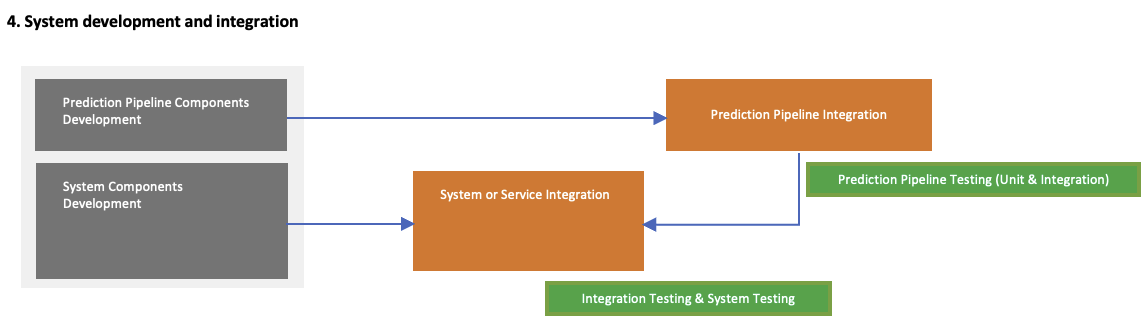


Figure 7 - Test activities in the phase of system development and integration.

In the system development and integration phase, the ML model is successively integrated into the software environment required for operation in production. As the first integration stage, we consider the integration of the model with software components that have a direct impact on the quality and performance of the model inference. This includes the integration of the model with the data sources for the inference (databases, user interfaces, sensors, etc), the data preprocessing components for the inference, and components that plausibilize or contextualize the result of the inference. We call the result of this integration the prediction pipeline. The model is then integrated with other system components until a complete system is available. The testing and quality assurance activities in this phase largely follow the established best practices of classical software testing.

*Prediction Pipeline Testing (Unit & Integration):* The prediction pipeline consists of the ML model, the software components that acquire, process, and feed data to the model, and the software components that directly interpret the model's prediction results. It can be assumed that especially the components of the prediction pipeline have a high degree of dependencies to each other. The test of these components takes place according to the strategies of the classical software testing by test of the individual components and the test of the integration as complete pipeline.

*Integration Testing & System Testing*: This activity aims to test all system components and its integration. Dependent on the definition of the system this varies from testing the prediction pipeline as mentioned above to arbirtrary integrations of the prediction pipeline as part of a complex ML-based system (e.g. an automated car or train). Integration and system testing is carried out based on a given integration strategy based on best practices and approaches well known in software engineering.

Table 4 provides an overview on the major work products of the system development and integrationphase, the related acceptance criteria and testing types and test objectives.

Table 1 Work products, acceptance criteria and test types of the system development and integration phase

|  |  |  |
| --- | --- | --- |
| **Work product/test item** | **Acceptance criteria** | **Test type/test objective** |
| Prediction pipeline | * ML model is correctly integrated in the prediction pipeline. * The prediction pipeline is correctly integrated with additional components e.g. safety mechanisms (safety cage, redundant models, plausibility checker etc.) | * Integration test (i.e., classical software testing) |
| ML-based system or component | * Prediction pipeline is integrated with the rest of the ML-based system. * Software-hardware embedding of the prediction pipeline and the ML-based system (model and data pre-processing or result preparation, GPU integration) | * Integration test (i.e., classical software testing) * System test (i.e., classical software testing) * Performance test for inference |
| Acceptance testing | * The ML-based system complies with stakeholder requirements | * Performance and stakeholder requirements testing * Testing the compliance with given rules and regulations |

## 8.5 The test items of the operation and monitoring phase

Ein Bild, das Text, Screenshot, Schrift, Rechteck enthält.

Automatisch generierte Beschreibung

Figure 8 - Test activities in the phase of operation and monitoring.

**For the operation and monitoring phase, the model is executed in its operating environment. Testing and monitoring activities must ensure that the model functions safely in the application context and is not outdated.** Depending on the assessed risk of the ML-based system during runtime, it is necessary to implement the execution of online testing (monitoring) of the system in operation. These tests go hand in hand with dedicated security and monitoring components that are supposed to identify corner cases and potential distribution shifts. As part of the system testing also the effectiveness of the online testing (monitoring) measures shall be verified.

*Acceptance Testing:* Acceptance testing for ML-based systems refers to the process of evaluating a trained machine learning model's performance when integrated within its software environment. Acceptance testing aims for determining whether an ML-based application meets the desired criteria and requirements established by stakeholders.

*Data Monitoring:* By monitoring the incoming data, it is possible to identify anomalies in the data stream, shifts in the data distribution and to detect concept drift. This allows to initiate special treatment of outliers and other anomalies and to re-evaluate assumptions on the data, update the model if needed, or trigger alerts for manual intervention.

*Prediction Monitoring:* Prediction monitoring enables you to track the performance of the model over time, detect any degradation in its predictive capabilities, and identify when it may need retraining or recalibration. By monitoring the technical model’s performance, it can be ensured that the ML-based application remains effective and allows to initiate timely adjustments if necessary.

*Business Monitoring:* Business monitoring aims to assess how well the ML-based applications are aligned with the business objectives and compliance rules. By monitoring business based key performance indicators (KPIs), it is possible to track the model’s performance in context of the associated business or application environment and allows to evaluate evaluating the economic impact and value generated by the ML-based applications. Moreover, it allows for proactive risk management, ensuring compliance with legal and ethical standards and maintaining trust among stakeholders and customers.

Table 5 provides an overview on the major work products relevant in the operations phase, the related acceptance criteria and testing types and test objectives.

Table 1 Work products, acceptance criteria and test types of the operation and monitoring phase

|  |  |  |
| --- | --- | --- |
| **Work product/test item** | **Acceptance criteria** | **Test type/test objective** |
| ML-based system or component | End user accepts the model in production | * User Acceptance testing (e.g. A/B testing, |
| ML-model | Model is free from drift | * Monitor data drift between the training and testing sets to ensure that the model is still accurate and reliable over time. * Monitor inference skew and bias |

# 9 Detailed test methods for testing ML-based systems (Uni Göttingen)

In this section, each test method is described according to the following structure:

* General definition of the test method
* How the test method works
* The type of issues the test method addresses (i.e., functional or non-functional issues)

## 9.1 Requirements-based testing (Gerhard)

The standard norm for testing is the fulfilled proof of all requirements in a requirement specification. ML-based systems, however, usually operate in open context, see chapter 2.2. An even approximately complete list of requirements will not exist. Therefore, explicit requirements are usually insufficient as a basis for testing. Nevertheless, a detailed analysis of the requirements specification should be the starting point for testing. Because in it the delimitation of the ML-based system is to be found, what it should do and where it is not responsible, and above all, in which environment it is to be used. Very helpful, if available, are detailed use cases [9], which narratively describe requirements and the possible interactions of the system with their intended environment.

Based on the requirements in the specifications, test cases must be systematically derived that can provide evidence of their fulfilment. The guiding question here is what can go wrong and will the identified, possible errors be completely covered up by tests.

Since ML-based systems usually infer probabilistically, an error cannot be proven by one counterexample, e.g. one wrong face recognition. Rather, it has to be checked during testing whether statistically the number of false results is significant. Only then is there a case of misbehaviour. This means, in particular, that in the case of ML-based systems, a single proof of an erroneous inference is not sufficient. If necessary, many similar tests must be defined and performed for a requirement so that a statistical evaluation of their results is possible.

### 9.2 Evaluation of the data (UNI Göttingen)

To evaluate the training data, its balance and completeness, it is essential to obtain a good overview of the various contexts in which the ML-based system is to be used. Therefore, it recommended to first extensively analyze and extend the given use cases. These virtual simulations can more precisely delineate the operation domain and identify further edge cases. Thus, they help to develop an ontology of the environment.

## 9.3 Risk-based testing (FhG)

**General definition**: Testing of safety-critical, security-critical or mission-critical software faces the problem of determining those tests that cover the essential properties of the software and have the ability to unveil those software failures that harm the critical functionality of the software. Even for "normal", less critical software, testing is usually done with severely limited resources and tight timelines, which means that testing efforts must be focused. This also involves more detailed testing of the functionality of a software, which are associated with a higher business risk. Both decision problems can adequately be addressed by risk-based testing which consider risks of the software product as the guiding factor to steer all phases of a test process, i.e., test planning, design, implementation, execution, and evaluation [33],[34], [35].

**How it works:** Risk-based testing is a pragmatic and often intuitively used approach [36] to focus test activities on those scenarios that trigger the most critical situations of a software system. It has become quite popular, and several approaches were developed in different context and application domain. See Erdogan et al. [37] for a comprehensive survey of risk-based testing approaches and [39] for with a systematic compilation of different approaches to risk-based testing in the context of IT security. In general, a number of different approaches exist for risk-based testing, with different emphases. A rough distinction can be made between risk-based test selection and risk-based test evaluation. Risk-based test selection addresses the problem that only a limited number of test cases can be realized or executed and that these test cases cover the use cases, functions or components to which the greatest risk is associated. A risk-based test evaluation, on the other hand, addresses the problem that the errors found during testing must be evaluated and, if necessary, a release can be made even with existing errors if these do not affect the critical functionality. The prerequisite for both approaches is a risk analysis. This can be formalized to varying degrees and ranges from an intuitive risk assessment by the tester to formalized and formal procedures with which an attempt can be made to describe risks qualitatively and quantitatively.

**Types of issues addressed:** Machine learning systems are systems that often operate in open environments, where it is fundamentally difficult to completely specify and delimit the often very extensive application environment. Strategies for risk-based test selection help to identify areas of the application environment that needs more extensive testing than others. Various factors influence the estimation of ML technology-related risks. Among others this includes risk exposure in the environment, severity of the hazard and statistical behavior of the ML-based component. Furthermore, machine learning is a stochastic approach with the consequence that the occurrence of errors usually cannot be completely avoided, and errors cannot be easily fixed. Therefore, ML-based systems enforce a paradigm shift that no longer focuses solely on the avoidance of individual software errors but take into account functional deficiencies and their relation to mission and business criticality. Thus, methods for risk based test evaluation are

Currently, there are only a limited number of risk-based testing approaches that specifically address machine learning. Some of the approaches are motivated by safety-critical applications in the field of mobility. Especially in the area of autonomous driving, there are a number of methods that deal with the identification and quantification of hazardous scenarios using various methods [40]. However, even though ISO 21448 recommends the combination of risk assessment and testing no systematic approach is yet described. In [41] Foidl and Felderer propose a risk-based data validation approach that tries to identify the risk of poor data quality for each feature used in training ML-based software systems. The risk of low data quality is calculated considering the importance of the feature for the overall system performance and the probability that feature is badly represented by the data. The latter is determined by assessing the data source quality, the data pipeline quality, and the occurrence of specific context-independent anomalies in the data. Schwerdner et. Al. [10] propose a risk-based approach to evaluate compare models for their robustness in a standardized way. The basis for the evaluation are so-called key risk indicators, which describe for concrete scenarios the probability of the occurrence of noise or corruptions as well as the errors resulting from these disturbances. The approach allows to compare models considering the errors weighted in terms of probability of occurrence and effect considering the special properties of the deployment environment.

### 9.5 Analytical Estimation (HWW)

Dynamic tests always check samples whose selection must also be justified. Therefore, in the field of conventional SW, complementary analytical estimations are usually expected. However, this is almost impossible for ML-based systems that use e.g., deep nets, since they are hardly analytically accessible so far with their multitude of parameters and nonlinearities. Current Explainable AI (XAI) research is investigating many different approaches here.

If the ML model has too many or too few parameters, new misbehaviour may occur. In the first case, the system learns to react convincingly in the training situations, but fails in new situations, (poor generalization, overfitting). In the second case, the system only moderately learns the training data, but usually behaves similarly in new situations (Underfitting). Thus, in novel situations, the system will react much more inappropriately or will consistently fail to exploit a possible mitigation potential. Indicator for overfitting or underfitting is monitoring and evaluating the learning process. If, in the further course of training, the loss values on the validation set are significantly higher than on the training data, this indicates overfitting. If, on the other hand, the loss values on both data sets remain equally bad despite extensive learning, this points to underfitting.

Analytically, behind the problem of over/under fitting lies the fact that the capacity of the network is not adequate to the problem (relevance of the model). For this purpose, the capacity or dimension of the selected ML model can be estimated analytically [12] and compared with the complexity of the learning data.

A neural network consists of a large number of nodes with simple activation functions and weights. Mathematically, it is a function that is supposed to represent e.g., a distribution function of empirical data. Depending on the number of nodes, layers, activation functions, more generally: architectures and weights, a variety of different functions can be represented by this construction, see [43]. With their help, the real distributions are during the training phase approximated. And as in the case of a Taylor or Fourier evolution of a function, the quality of the approximation by neural networks depends on the architectures and the number of weights used. Insufficient approximation can then lead to data that cannot be separated by the function (insufficient separability), which is noticeable as underfitting. On the other hand, the training data are always only samples of the real data, comparable to the interpolation points of functions for extrapolation. If now the approximation to this random selection is too exact, the extrapolation is insufficient, the generalization is insufficient, i.e., we have overfitting.

In order to avoid underfitting and overfitting it is necessary to take a sufficiently large class of functions for the approximation, which on the other hand is also not too large. To describe the power of function classes, a criterion was developed in the context of statistical learning (Vapnik et. al.), with which the capacity of these sets of functions can be captured, the Vapnik-Chervonenkis dimension [44]. Depending on the complexity of the data space and the used architecture resp. number of weights this dimension can be estimated. It then provides an analytical assessment of the quality of the ML system with respect to overfitting and underfitting. For recent estimates see [45].

On the other hand, the work of Madry et. al, [15] shows that as the generalizability of a network increases, its robustness can decrease.

A very general analytical criterion to estimate robustness is the Lipschitz constant, see [46]. It describes how much a function value can change maximally, if the input values differ only slightly, formally

|F(x)- F(y)| < L |x-y| for all x,y, L -Lipschitz constant.

Basis for this a distance measure on the data space as well as on the image space. If, as in classification problems, the later is discrete, the values of the penultimate layer, which usually represent the probabilities of the different discrete image values, can also be considered. An estimation of this constant then gives a measure for the general robustness of the networks, see e.g. [47].

## 9.6 Search-based testing (Großmann)

**General definition:** Search-Based Testing (SBT) is the application of optimizing search techniques to solve software testing problems. capabilities. Among others SBT is used to generate test data, prioritize test cases, minimize test suites, optimize test oracles, increase test coverage, and validate real-time properties of software. The search algorithms can be guided by different criteria, such as code coverage, requirements coverage, or fault-detection. In general this may include random search, to randomly generates test inputs and evaluates their effectiveness in revealing faults, genetic algorithms that generate a population of test cases, evaluate their fitness (based on a defined objective function), and use selection, crossover, and mutation operations to evolve the population over multiple generations, particle swarm optimization, where swarm of particles moving through the input space and the swarm collectively explores the space to find promising solutions. The effectiveness of search-based testing depends on factors such as the quality of the search algorithm, the representation of test inputs, and the defined objective functions. It is often used in combination with other testing techniques to complement and enhance the overall test coverage and fault detection capabilities.

**How it works:** The key idea of SBT in ML is to leverage search algorithms to explore and navigate the various spaces associated with machine learning models, parameters, data, and configurations to identify potential model performance issues, robustness issues, and efficiency issues. SBT can be applied as long a continuous optimization function could be found. It supports activities like data preparation, feature selection and extraction, model evaluation, adversarial testing and in reinforcement learning.

**Types of issues addressed:** Since both training an ML model and search-based testing are optimization processes, SBT can be applied to machine learning in several ways to improve testing and validation of ML models or other artifacts in the ML life cycle. For example, machine learning models often have numerous hyperparameters that need to be set appropriately for optimal performance. Search-based testing can be employed to explore the hyperparameter space efficiently, finding combinations that yield the best performance metrics or minimize errors. Moreover, SBT can help identify the best model for a specific task by comparing different algorithms or configurations. The search process can consider factors like accuracy, computational efficiency, and model complexity to select the most suitable option. By exploring the feature space, SBT can be used to determine the most relevant features or feature combinations for a machine learning model, identify subsets of features that result in improved performance or reduced dimensionality and leading to more efficient and accurate models. In addition, SBT ca be used for synthetic data generation, augmentation, and data space exploration, ensuring better coverage of input variations and uncovering potential edge cases. [14]. Finally, SBT can be employed to search for such adversarial examples efficiently [15], helping identify weaknesses and improving the robustness of the model. Finally, SBT techniques can also be applied to reinforcement learning settings e.g., to optimize the agent's behavior or policy.

## 9.7 Combinatorial testing (Jürgen)

**General definition:** The principle behind combinatorial testing is based on the observation that many defects or failures in software systems are caused by interactions between different input parameters rather than by individual parameters in isolation. By testing a range of parameter combinations, i.e., combinations that each include two, three, or some other number of parameters, the technique can effectively detect a large portion of the errors arising from interaction effects. The choice of the appropriate value of for parameter combinations depends on factors such as the complexity of the system, the number of input parameters, and the available resources. Pairwise testing (2-wise) is often used as a starting point, as it provides a good balance between coverage and efficiency. It covers interactions between pairs of parameters, which tend to be the most critical in terms of defect detection. However, higher values of "n" can be chosen when there are specific concerns about interactions involving more than two parameters.

**How it works**: To generate test cases for n-wise testing, various algorithms and tools are available that employ combinatorial design theory or optimization techniques. These tools generate a minimized set of test cases that cover all possible combinations of n parameters with minimum redundancy, ensuring comprehensive coverage while minimizing the testing effort. However, the application of combinatorial testing in ML is even for small input spaces challenging since the number and possible valuations of the individual input parameters are too large.

However, there is a number of potential application scenarios when the input space is subdivided by a systematic classification approach that reflects for example typical situations, risk areas, possible sources of noise and other influences at first, and different aspects thereof in a concise refinement, leading to model or ontology that covers an abstract representation of the input space by covering various viewpoints. Based on such a model, combinatorial testing provides a means to get a systematic test coverage following an equal distribution over the different aspects represented by the model. Providing a weighted model that, besides the manifestation of the object and features of the domain, also specifies the frequency of their occurrence, the associated risk, etc., combinatorial testing could even provide a good estimation of the required distribution of the training and test data.

**Types of issues addressed:** Combinatorial testing could be used to select and generate test and training data for model testing. In [42] Gladisch et al show how combinatorial testing can be used to generate test, training and validation sets based on a domain model. In particular this approach is considered useful for systematic generation of synthetic data.

## 9.8 Metamorphic testing (UNI Göttingen)

**General definition:** Metamorphic Testing (MT) is a property-based software testing approach which offers the possibility of alleviating the oracle problem and thus can be used to test non-testable systems. The general idea of MT is to apply a set of predefined Metamorphic Relations (transformations or metamorphisms) to a source test case in order to generate follow up test cases which are tested against the system. If the output of the follow-up test cases violates the defined metamorphic relation, then the system can be considered as defective.

**How it works:**

Definition of Metamorphic Relations: In MT, the first step is to identify Metamorphic Relations (MR) that define how the input and output of the system should change in response to a specific transformation. For example, if the ML model is trained to recognize handwritten digits, an MR could be that flipping the image horizontally or vertically should not change the predicted digit.

Generation of Test Cases: the next step is to apply the defined MRs to the original input data in order to generate new test cases (transformed version of the original input data)

Comparison of Outputs: In this step, we compare the output of the original input data and the transformed versions. If the output of the system is consistent for all versions of the transformed input data, then the system passes the test. However, if the output of the system is found to be inconsistent for any of the transformed versions of the input data, then the test fails, indicating that the system has a bug or a number of problems. For example, if an ML model train to recognize handwritten digits is unable to classify correctly a flipped handwritten digit, then this is an indication of a potential problem [2].

**Types of issues addressed:**

Metamorphic testing is primarily a useful technique for addressing functional issues. However, it may also be useful for the detection of non-functional issues such as reliability and performance-related issues.

It is worth noting that a passed MT does not necessarily guarantee the correctness of the system. For instance, a metamorphic relation applied to a mislabeled image will pass a Metamorphic Test without exposing the mislabeling.

[1] M. D. Davis and E. J. Weyuker, “Pseudo-oracles for non-testable programs,” *Proceedings of the ACM ’81 conference on - ACM 81*, 1981. doi:10.1145/800175.809889

[2] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes, “A survey on metamorphic testing,” *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016. doi:10.1109/tse.2016.2532875

## 9.9 Differential testing (Universität Göttingen)

**General definition:**

Differential Testing also known as “Back-to-Back Testing” is a testing technique used in software development that involves comparing the output of two versions of a program that ought to produce the same results. The purpose of Differential Testing is to detect differences or discrepancies between the two versions of the program, which can be indicative of bugs or unusual behaviour [1].

**How it works:**

In the context of machine learning, Differential Testing involves comparing the output of multiple implementations of the same learning algorithm which have also been trained on the same training data[2]. If there is a difference between the results, then presumably one or both implementations have a bug. For instance, if a Graph Neural Network (GNN) model with the same network and weights behaves differently when running on two different GNN implementations (such as PyTorch and TensorFlow), it is likely that one of the implementations is incorrect, even if the expected output is unknown.

A drawback of Differential Testing is its resource inefficiency due to the multiple system runs, and its susceptibility to errors as the same errors may occur in various implementations of the system under test [3].

**Types of issues addressed:**

Differential Testing may be used to address both functional and non-functional issues. Functional issues may include cases where one implementation of the model produces incorrect predictions compared to the other implementation, while non-functional issues may include cases where one implementation of the model takes longer to produce results or uses more resources than the other (i.e., this may be difficult to compare across platforms, scaling may need to be applied).

In conclusion, Differential Testing is an important technique in machine learning testing that can help detect bugs and unexpected behaviour in an ML model by using one implementation of the ML model as a pseudo-oracle for the other.

[1] W. M. McKeeman, “Differential Testing for Software,” *Digit. Tech. J.*, pp. 100–107, 1998.

[2] C. Murphy, G. E. Kaiser, and M. Arias, “An Approach to Software Testing of Machine Learning Applications,” *International Conference on Software Engineering and Knowledge Engineering*, 2007.

[3] D. Marijan and A. Gotlieb, “Software testing for Machine Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, pp. 13576–13582, 2020. doi:10.1609/aaai.v34i09.7084

## 9.10 Adversarial Attacks (Universität Göttingen)

**General definition:**

Adversarial Attacks refer to the subtle modification of original inputs to a trained machine learning model to cause it to make incorrect predictions or decisions. These attacks are typically carried out by adding small, carefully crafted perturbations to input data that are almost imperceptible to human observers but can significantly affect the output of the model. Adversarial Attacks are a growing concern in the field of machine learning, as they can potentially compromise the security and reliability of machine learning systems.

**How it works:**

In the context of image classification, Adversarial Attacks work by discovering a slight modification that when applied to an original image, leads the model to inaccurately classify it, while still being correctly classified by the human eye [1]. For instance, for a given input image x, the objective is to find the smallest possible modification η such that the resulting altered image (i.e., adversarial example) x’ = x + η is misclassified. Adversarial attacks can be categorized as either targeted or untargeted. In a targeted attack, the adversary aims for the modified image x’ to be classified as a specific class t, whereas in an untargeted attack, the adversary’s objective is for the modified image x’ to be classified as any class other than its correct class [2]. To mitigate this risk, Adversarial testing otherwise known as adversarial training is performed by incorporating identified adversarial examples and the corresponding ground truth labels into the training data in order to ensure that the model is trained to correctly identify them [3].

**Types of issues addressed:**

Adversarial Attacks can address both functional and non-functional issues in machine learning models. Functionally, these attacks can expose weaknesses in a model's decision-making process, revealing its vulnerabilities to malicious inputs. Non-functionally, Adversarial Attacks can also help to evaluate the robustness and reliability of machine learning models, as well as to identify potential areas for improvement in their design and implementation.

[1] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, “Detecting Adversarial Samples from Artifacts,” *ArXiv*, 2017.

[2] J. Lin, L. L. Njilla, and K. Xiong, “Secure machine learning against adversarial samples at Test Time,” *EURASIP Journal on Information Security*, vol. 2022, no. 1, 2022. doi:10.1186/s13635-021-00125-2

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” *CoRR*, 2014.

## 9.11 Exploratory testing (Jürgen)

<TBD> Systematic testing approaches are used for this purpose, which, for example, divide the input data of a software system into classes and test selected representatives of each class. Furthermore, testing can contribute to a better understanding of an unknown system. The idea of exploratory testing describes testing as a creative act in which the tester learns new properties of the SUT by performing tests and can incorporate them into new tests. However, software testing cannot prove that the system is free from faults.

## 9.12 Probabilistic testing

<TBD>

## 9.13 Testing with failure models

<TBD>

## 9.14 Diversifying test

<TBD>

Back to back test, regression test

## 9.15 Reviews

Reviews, as a software quality assurance method, involve a systematic examination and assessment of software artifacts or deliverables by a group of individuals with relevant expertise. The goal of reviews is to identify defects, improve the quality of the software, and ensure compliance with standards, guidelines, and requirements.

**How it works:**

**Types of issues addressed:**

In traditional software engineering requirements, design, code, user interface

In machine learning reviews can address diverse ml artifacts like data, labels, hyperparameters and the model itself as well as documentation of data, model and the trainings process. Morevoer reviews can target different quality attributes like performance, bias and fairness, Explainability and interpretability, compliance

In general, reviews help identify defects or issues early in the development process, reducing the cost and effort required to fix them later. Reviewers can share their expertise and knowledge, improving the overall quality of the software and enhancing the team's understanding of the project. promote collaboration and foster a learning culture within the development team, leading to continuous improvement and knowledge transfer. Finally, reviews help ensure that the software artifacts comply with industry standards, guidelines, and regulatory requirements.

## 9.16 Static analysis

In special cases, e.g. for safety-critical situations, dynamic tests with their random sampling character are not sufficient. Verification techniques might then be used.

Under specific conditions, e.g. in the case of neural networks with discrete outputs and monotonic activation functions, it can be shown that their function can be modeled by linear arithmetic expressions, see e.g. [48]. If one now formulates the specific input situation and the undesired output also as linear arithmetic formulas, then constrained-based verification algorithms (SAT solvers) can be used to rule out the occurrence of a negative output.

For networks with ReLu activation functions, the simplex algorithm was extended for this purpose, see [49].

The networks in ML systems can become very complex. Then help may be abstraction techniques in which numerical values are replaced by simple geometric sets. Algorithms estimates its way through the net and help to estimate the resulting image sets, see Zonotope Abstraction, see [48].

In practice, however, the networks are usually so complex that these techniques can hardly be used successfully, despite the best optimizations.

# 10 Workflow and process aspects of testing ML-based systems

## 10.1 Test Management for testing ML-based systems

TBD.

## 10.2 Dynamic test process for testing ML-based systems

### 10.2.1 Test planning phase

*Roughly speaking, the test planning phase serves to define the quality objectives, determine the test items and set up a test strategy that serves to test the desired quality objectives in a meaningful way. Afterwards the entire test process is planned in its technical, temporal, and monetary aspects, taking into account the available resources.*

A test strategy describes which parts of the system are to be tested with which intensity, using which test methods and techniques, using which test infrastructure and in which order.

Testing ML-based systems places some special challenges on the test planning phase.

**Challenge 1: Selection of appropriate quality and test items**

Since ML-based system slightly differ in terms of engineering as well as operation, the test process must address additional test itemives, that are often not addressed in classical software testing. Besides coverage of the relevant functional aspects of the application context including standard cases/scenarios, all critical corner cases/scenarios as well as all defined non-functional properties like security, robustness, performance etc., testing ML-based systems need to reveal

* data and labelling errors that lead to critical functional failures
* software failures that undermine critical functionality during model training and model inference
* unused or unintended decision capabilities of a model
* bias and noise in decision processes
* known vulnerabilities and failure modes of the technology used eg. in DNNs/CNNs

**Challenge 2: Determining all relevant test items and the corresponding test procedures**

To comprehensively test ML-based software systems, several new test items must be considered that are given little to no attention in classic software. These test items are:

* data and labels
* hyperparameters
* loss function
* optimiser
* training KPIs and acceptance criteria
* network architecture and additional design decision defining basic model properties
* the ML-Model including the software implementation of the models’ internal behaviour and all parameter settings
* the ML-Framework including the used libraries and algorithms
* data pre-processing software during engineering
* additional components that serve a proper integration of the ML-model including safety mechanisms (safety cage, redundant models), model and data pre-processing or result preparation, GPU integration.

**Challenge 3 Definition of an appropriate integration and test procedure.**

ML-based systems are complex entities with high dependencies. Thus, the quality of an ML-based decision system is not only based on the performance of the ML model, but also on

* the performance of the data pre-processing chain including all the required sensors and data fusion components,
* the software that interprets the output of the ML model, processes it for humans and/or translates it into actions, and
* the seamless interaction of all these components.

In addition, the quality of the target system is dependent on the training data, data preparation, and training infrastructure. Thus, a systematic test approach does not only target the system and its integration, but also the entire data acquisition and training infrastructure. If we take this into account, the test levels of classical software testing can be extended as follows.

* data pipeline testing
* training pipeline testing
* data and data integration testing:
* component testing: ML-Model, data pre-processing, decision making
* integration testing: Model in data pre-processing chain, Model in data pre-processing chain + decision making, ML-model subsystem with safeguarding
* system testing: Entire system in test environment
* acceptance testing: Entire system in operational environment
* runtime testing

### 10.2.2 Test design & analysis phase

*The test design and analysis phase serve to implement the test items defined in the strategy in a meaningful way. This includes the identification of the abstract tests, the definition of suitable coverage and completeness measures and the specification of suitable procedures and frameworks for the automation of the tests.*

**Challenge 1: Identification of appropriate data testing procedures**

Due to the high importance of data for the performance of a ML model, both the data, its origin, its storage, and preparation must be systematically tested and reviewed. In this context we distinguish between testing the data acquisition, preparation and storage infrastructures and testing the data and data quality itself.

Testing the data acquisition, preparation and storage infrastructures mainly addresses aspects of infrastructure testing like data base testing, testing the underlying communication and computation platforms regarding performance and availability, and the data processing infrastructures that allow for data preparation and refinement. The test approach must consider that these infrastructures are often dealing with big data that is, most of the processes are highly automated and require a high degree of availability and scalability that poses special requirements on hardware and software solutions with corresponding challenges for testing (see [16][17]).

According to L.P. English [18]data quality can be subdivided into three aspects, which can be considered independently of each other.

* Data definition and information architecture quality describes the quality of the data specification based on the application context.
* Data content quality describes the inherent quality characteristics of the data such as correctness of data values, completeness, unambiguity, freedom from errors, etc.
* Data presentation quality describes how the data can be made available appropriately quickly, in a suitable format, and with a reasonable amount of effort.

Data quality dimensions are attributes of data quality that, if measured correctly, can describe the overall level of data quality. The identification of relevant quality dimensions forms the basis for the assessment and subsequent improvement of data quality. The quality dimensions are usually highly context-dependent, and their relevance and importance can vary depending on the organization and data type. The most common, i.e., the most frequently cited dimensions in the literature, are completeness, timeliness, and accuracy, followed by consistency and accessibility [19].

Overall, assessing data quality for ML applications is a complex task. Current best practices suggest that more data and better models provide better results.

* Poor data quality can cause significant problems in both ML model building and big data applications.
* Certain systematic preprocessing operations on the data help these models achieve higher effectiveness.
* While traditionally data quality is assessed before the data is used, in the machine learning context quality can be assessed both before and after the model is built.
* Data quality can be assessed before the learning process along the data and its compilation processes and after the learning process along the performance of the ML model.
* The data quality is evaluated along different quality attributes, so that systematic evaluation criteria for the data quality can be established.

To date there are no testing approaches that directly address the issues from above in a systematic and automated manner.

**Challenge 2: Identification and selection of appropriate tests for complex/open world scenarios**

Testing machine learning suffers from a particularly difficult form of the oracle problem. While classical systems are usually fully specified, machine learning systems are designed to provide meaningful answers to questions for which there is not yet an answer known [1] (Zhang et al.). Training ML models typically aims to achieve good performance on training data while being able to generalize well to unseen, new data. For the models to learn the underlying function from the data provided to them, that data must sufficiently capture the features of the real-world problem. If incomplete, outdated, or irrelevant data are provided to the model, the model will not generalize towards unseen data.

The problem for testing then consists of defining suitable criteria for defining the completeness of the data for a partially unknown range and to generate test cases that systematically represent the entire input range. In addition, the test cases must be stored with suitable expected values that allow a systematic evaluation of a test run. This special form of the Oracle problem known from testing prevents a scalable test data generation. Solution approaches, such as metamorphic testing [32], are not yet able to realize the necessary scalability and efficiency required for a comprehensive testing approach.

**Challenge 3: Dealing with ML-specific failure modes**

Since ML and ML-based systems show significant differences to classical software engineering, testing processes may fail if they do not address failure modes that are specific for ML-based systems. These failure modes include bias, non-determinism, lack of robustness, and lack of transparency and understandability.

**Decision bias:** Bias in machine learning is a type of error in which certain elements of a dataset are weighted and/or represented more heavily than others. A biased dataset does not accurately represent the intended use case of a model, leading to biased results, low accuracy, and analytical errors. Bias can occur in several different areas, from human reporting and selection bias to algorithmic and interpretation bias. Sampling bias, for example, occurs when a dataset selected for training does not reflect the realities of the use case (e.g., when facial recognition relies significantly on data from only one population group e.g., men, women, Europeans). Exclusion bias most often occurs in the pre-processing phase of the data. It is often caused by the deletion of valuable information that is considered unimportant e.g., the deletion of a relevant feature that has not been recognized or that has been considered as unimportant. Measurement bias occurs when the data collected for training is different from the data collected in the real world, for example, when different sensors are used to record the training data as with the production data. Measurement bias can also result from inconsistent label assignment during the data labelling phase of a project. Finally, observer bias also known as confirmation bias, is the effect of seeing what you expect or want to see in the data during manual data selection and labelling processes.

**Probabilistic nature and non-determinism:** ML-based software, even if it has some fundamentally deterministic properties, is not necessarily stable with respect to the environment and environmental changes. Moreover, the training process itself is often nondeterministic and thus difficult to reproduce. Non-determinism in the training phase arises from the random initialization of model parameters, the stochastic selection of training data (e.g.  mini batch sampling), and the use of stochastic functions in the optimization process. Non-determinism in the operation phase may arise using stochastic activation and weight functions. Moreover, neural networks are typically trained on graphics processing units (GPUs), which, under certain experimental conditions, yield nondeterministic outcomes for floating point operations.

**Missing robustness:** Robustness is the ability of a computer system to deal with erroneous input and to handle errors during execution. An ML model is considered robust if small perturbations in the input space yield only small perturbations in the output space. Since ML has been shown to be especially vulnerable against so called adversarial examples and against distributional shift, it can only be considered robust under certain circumstances.

An adversarial example is an input to a neural network that has been modified in such a way that it alters the output of the neural network, even though a human would still recognize the original class. In the extreme case, the modified input is indistinguishable from the original input for a human. Distributional shift describes a difference between the test and training environments [Ref 1]. Such distributional differences can be considered as gaps in the representation of reality and are a general problem in designing ML applications to be used in real-world applications. If the perceptual or heuristic inference processes of such a model have not been adequately trained to the correct distribution or the distribution of the environment changes in operation, the risk of unintended and harmful behaviour increases significantly.

**Lack of transparency and understandability:** Neural networks function as black box systems. Instead of humans explicitly coding the system behaviour with conventional programming, in ML the computer program learns based on many examples that represent the mapping of the input data to the desired output. Transparency in AI is generally referred to as explainability, which includes both interpretability and confidence in the system and its genesis[29][30]. While interpretability is the degree to which a human can understand the cause of a decision [31], confidence in a system is gained by understanding the system itself, its operational environment as well as the development of the system.

A challenge regarding testing arises from the dependence on a system that not even the developers and testers really understand. To gain confidence and certainty regarding elemental quality properties of neural networks, it is essential to enable at least a certain degree of human interpretability and understandability.

**Challenge 4: Definition of appropriate coverage and completeness criteria**

Due to the lack of logical structures and system specification, it is still unclear how evidence regarding test completeness could be provided for ML-based systems especially for those with DNN components. To date, there are several proposals that combine systematic testing of ML-based systems with coverage criteria related to the structure of DNNs. These include simple neuron coverage by Pei et al. [23], which considers the activation of individual neurons in a network as a variant of statement coverage. Ma et al. [22] define additional coverage criteria that follow a similar logic to neuron coverage and focus on the relative strength of the activation of a neuron in its neighbourhood. Motivated by the MC/DC tests for traditional software, Sun et al. [24] proposes an MC/DC variant for DNNs, which establishes a causal relationship between neurons clustering i.e., the features in DNNs. The core idea is to ensure that not only the presence of a feature, but also the combination of complex features from simple feature needs to be tested. Wicker et al. [25] and Cheng et al. [26]refer to partitions of the input space as coverage items, so that coverage measures are defined considering essential properties of the input data distribution. While Wicker et al. discretizes the input data space into a set of hyper-rectangles, in Cheng al. it is assumed that the input data space can be partitioned along a set of weighted criteria to describe the operating conditions. Finally, Kim et al. [21] evaluate the relative novelty of the test data with respect to the training dataset by measuring the difference in activation patterns in the DNN between each input. A good summary of the current state of the art regarding coverage criteria for testing DNNs can be found in [20]. In addition, the work of Dong et al [27] claims that there is only a limited correlation between the degree of different kinds of neuron coverage and the robustness of a DNN, i.e., improving the degree of simple neuron coverage measures does not significantly contribute to improving the robustness. However, in their study, Dong et al. did not analyse the effect of more complex coverage approaches (e.g., feature coverage and the MC/DC variant for DNNs) as well as coverage approaches that address the partitioning of the input data space.

### 10.2.3 Test Implementation & execution phase

*During the implementation and execution phase test cases are created and executed. Test cases should be based on the objectives and requirements identified during the planning and analysis phase. During the execution, the test team performs all tests. The deviations are logged, and defects are identified. Deviations are measured as the difference between actual and expected test results.*

**Challenge 1: Synthetic test data generation**

ML systems process a wide variety of data. These range from simple tabular data to complex data streams (images, movies, radar or lidar data), such as those processed in ML-based perception systems. To be able to test such systems and to make the necessary large amounts of data available in sufficient diversity, data will have to be synthetically generated. The more complex the input data, the more complex is the process of data generation. For example, the creation of synthetic film sequences is significantly more complex and resource-intensive than the provision of simple numerical quantities.

**Challenge 2: Achieving the necessary degree of automation and scalability.**

The complexity and uninterpretability of DNNs lead to the fact that manual testing approaches are not sufficient to perform a comprehensive quality assurance of a DNN.

To cope with the complexity of the applications and to achieve consistent results in repeated tests a high degree of automation is required. Automation should encompass all necessary activities of the testing process, starting with test case identification, test data generation, test execution, and final test evaluation. Similar, to the training of an ML model, such an automated testing approach relies on a larger technical infrastructure that realizes automation in a in a trustworthy and reliable manner.

However, generating test cases automatically is still a challenge. For instance, studies [85, 86] claimed that the test cases generated by an automated testing tool may not cover all real-world cases. (Zhang 2020)

### 10.2.4 Evaluating exit criteria and reporting phase

*The test evaluation and reporting phase is used to evaluate the test execution against the defined and agreed exit criteria. Based on this evaluation, a decision can be made as to whether enough tests have been performed to achieve the quality objectives defined in the planning phase. The result of the test evaluation is then documented and summarized in a form that can be understood by all relevant stakeholders.*

**Challenge 1: Define and apply appropriate end-of-test criteria and validation metrics.**

The interpretation, aggregation and evaluation of individual test results and the evaluation of the entire test process for ML-based systems can differ greatly from the procedures that are established for classical software systems. On the one hand, completely new test procedures have to be taken into account due to the consideration of data as a decisive quality factor, and on the other hand, the specific characteristics of an ML-based system, especially with regard to its failure characteristics, lead to different evaluation approaches.

On the one hand, DNNs in particular feature a complexity that is not reached by classic software. While it is possible to trace failure modes back to individual errors in classical software systems, this is much more difficult in ML-based systems. The high number of parameters, hyperparameters and optimization decisions makes it almost impossible to identify wrong parameters as the cause of a concrete failure mode.

Additionally, when considering different quality properties, it is important to keep in mind that there are dependencies between these properties, so that improving the KPIs for one property will worsen the KPIs of another property.

Risk-based testing approaches are basically able to relate variable quality properties of a system to the risks to the financial and fundamental risks of an application. An end-to-end approach on how to comprehensively apply risk-based testing in the context of ML systems has been sparsely explored.

**Challenge 2: Communicate test status and evidence on quality in a comprehensible and trustworthy way**

Test reports are designed to enable managers and users of software products to assess and understand the quality and risks of a software product in its application. To this end, the tests, their results, and the metrics used to demonstrate the performance of an ML-based system must be expressed in terms of their impact on the application domain in an understandable way. This is particularly important when it comes to assessing interconnected quality properties between which there may be a conflicting objective.

# 11 Testing in the ML Life Cycle

## 11.1 The ML Life Cycle

Ein Bild, das Diagramm enthält.

Automatisch generierte Beschreibung

Figure 4 The MLOps life cycle

## 11.2 Plan

***Purpose:*** *This phase serves to develop a basic understanding of the problem to be solved by ML as well as the data required for it. Developers and spzeificators decide which features are feasible with machine learning and which can be useful for a given existing or a new product. Importantly, it is at this stage that they decide what types of models are best suited for the given problem and what data is needed to successfully learn the desired features.*

* Testing requirements and KPIs

## 11.3 Data Engineering

*This phase aims to identify the right data sets in the right distributions, collect the data in such a way that the model output can be delivered as efficiently as possible, enrich the data through labeling, store the lineage of the data, verify the quality of the labeled and prepared data, establish specific metrics to measure the quality of the data, store and analyze the data.*

* Testing the data sources and their reliability
* Testing the data preparation and training pipeline (training phase)
* Testing the data

## 11.4 Code

***Purpose:*** *This phase aims to develop the source code that is required for setting up the model and related components. This includes, among other things, all components that form the data preparation and decision pipeline during deployment. The individual components, the software implementation of the model and the integration of all components into a functional unit must be realized and tested.*

* Testing the decision pipeline (in integration)

## 11.5 Model Engineering

***Purpose:*** *This phase aims to parameterize and train model variants based on the available data and their labels by considering all related requirements, and to identify and select the most appropriate model. In practice, training runs are performed with different model architectures and initial parameters (hyperparameters) and the results are compared. All resulting models are benchmarked, evaluating their properties in terms of accuracy and generalizability. The best model(s) is/are selected and passed to the subsequent process for further V&V. For modeling, predefined ML frameworks are usually used to support the developers in creating the model code and realizing the algorithms.*

* Testing the model architecture and other design decisions
* Testing the model pipeline
* Testing the training assumptions (Hyperparameter settings, algorithms etc.)
* Testing the model software and third-party libraries

## 11.6 Model V&V

***Purpose:*** *In this phase, the model is fully validated and verified. The aim is to assess the extent to which the model fully meets the functional and extra functional requirements given. In addition to accuracy and generalizability, scalability, complexity, robustness, fairness and resource requirements in particular play a crucial role in the evaluation of a model. The model is checked against all relevant performance characteristics and KPIs. Finally, it is decided at this point whether the model(s) at hand are suitable for integration into the software stack or need to be improved via further iterations.*

* Testing the functional aspects of the ML-model
* Testing extra-functional aspects of the ML-model e.g. robustness, fairness, transparency, security, scalability

## 11.7 Test

***Purpose:*** *The selected model must be tested and evaluated in its deployment environment. For this purpose, the model must be integrated with the software and hardware of the target platform and systematically tested. The evaluation focuses on the safe functioning of the model in interaction with the necessary software and hardware as well as the interaction with additional safety components and functions, such as watchdogs for identifying dangerous situations and performing plausibility checks, redundancy for safeguarding the overall functionality, etc.*

* Model integration testing
* SW/HW integration testing

## 11.8 Integration

***Purpose:*** *In this phase, the model is transfered in a real-world setting or integrated into the application/edge device it was developed for. Most organizations typically follow stringent procedures, or even provide standard templates for building a complete release package. Testing happens throughout this process, starting with testing all input CIs, to testing and rehearsing the services before they are deployed live.*

* Testing end to end functionality

## 11.9 Deploy

***Purpose:*** *In this phase, the release package is deployed to the live environment, beginning when change management authorizes the release package to be deployed to the target environments. Afterwards operators deploy the application to a production environment. They may also perform maintenance tasks on the application. There are different levels of deployment for models. In shadow deployment, the model results are not used directly. Deployment occurs in parallel with another model or the final decision is made by a human, regardless of what the model predicts. Usually this is done to determine how well the model is performing. In canary deployment, the model is presented with only a small portion of the data on which it is allowed to make decisions. Depending on how the model performs, the data is gradually increased, or the model is withdrawn, and adjustments are made. At Blue-Green deployment, the traffic is gradually changed from an old version (blue version) to the new version (green version). In this way, any kind of downtime is avoided and in case of errors, the application can be easily reverted to the previous stable version or the blue version.*

* Acceptance testing

## 11.10 Operate

***Purpose:*** *This phase is organizing the operation of the ML-based application. It aims to minimize or eliminate downtime for your end users through efficiently managing hardware and software changes.*

* Runtime testing

## 11.11 Monitor

***Purpose:*** *Monitoring of the ML-based application and its application environment is the final phase in the MLOps cycle. It builds on the customer feedback by collecting data and providing analysis on customer behaviour, application performance, and defects. Regarding the ML-models, monitoring includes the predictive quality, the inference performance, the occurrence of special kinds of failure, and certain kinds of drift. Especially in highly automated MLOps processes, this includes not only data about the application and the model, but also data about the entire infrastructure, i.e. the compute environment, the data and modelling pipelines, trying to identify bottlenecks in the process of continuous training and the provision of new models and application updates. The monitoring data is used to take automated decisions like e.g., retraining and is fed back to the product manager and the development team in a structured and preferably automated way.*

Annex A:  
Title of annex

Annex:  
Bibliography

1. Zhang, J. M., Harman, M., Ma, L. & Liu, Y. Machine Learning Testing: Survey, Landscapes and Horizons. arXiv:1906.10742 [cs, stat] (2019).
2. Humbatova, N. et al. Taxonomy of real faults in deep learning systems. in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering 1110–1121 (ACM, 2020). doi:[10.1145/3377811.3380395](https://doi.org/10.1145/3377811.3380395).
3. Poddey, A., Brade, T., Stellet, J. E. & Branz, W. On the validation of complex systems operating in open contexts. arXiv:1902.10517 [cs] (2019).
4. M. Pol, T. Koomen, und A. Spillner, Management und Optimierung des Testprozesses: ein praktischer Leitfaden für erfolgreiches Testen von Software mit TPI und TMap, 2., Aktualisierte Aufl. Heidelberg: dpunkt-Verl, 2002.
5. V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, und P. Tonella, „Testing machine learning based systems: a systematic mapping“, Empir Software Eng, Bd. 25, Nr. 6, S. 5193–5254, Nov. 2020, doi: [10.1007/s10664-020-09881-0](https://doi.org/10.1007/s10664-020-09881-0).
6. L. Myllyaho, M. Raatikainen, T. Männistö, T. Mikkonen, und J. K. Nurminen, „Systematic literature review of validation methods for AI systems“, Journal of Systems and Software, Bd. 181, S. 111050, Nov. 2021, doi: [10.1016/j.jss.2021.111050](https://doi.org/10.1016/j.jss.2021.111050).
7. Gal, Yarin. Uncertainty in Deep Learning, University of Camebridge, October 13th, 2016
8. M. Borg, „The AIQ Meta-Testbed: Pragmatically Bridging Academic AI Testing and Industrial Q Needs“, arXiv:2009.05260 [cs], Sep. 2020, Zugegriffen: Okt. 13, 2021. [Online]. Verfügbar unter: <http://arxiv.org/abs/2009.05260>
9. Jörn Müller-Quade et al., Sichere KI-Systeme für die Medizin, <https://www.plattform-lernende-systeme.de/files/Downloads/Publikationen/AG3_6_Whitepaper_07042020.pdf>
10. Paul Schwerdtner et.al., Risk Assessment for Machine Learning Models, arXiv:2011.04328v1
11. Michael Felderer et. al., A taxonomy of risk-based testing, arXiv:1912.11519v1
12. Vladimir Vapnik et.al, Measuring the vc-dimension of a learning machine, Neural computation 1994.
13. Matthias Grochtmann, et. al., Classification Trees for Partition Testing, Software Testing, Verification & Reliability. 3, Nr. 2, 1993
14. Ian J. Goodfellow et. al., Generative Adversarial Nets, arXiv:1406.2661v1
15. Madry et.al. Adversarial Examples Are Not Bugs, They Are Features, arXiv:1905.02175v4
16. Arniban Charkroboty et. al., Adversarial Attacks and Defences: A Survey Xiv:1810.00069v1
17. Steidl, Monika, Ruth Breu, und Benedikt Hupfauf. 2020. „Challenges in Testing Big Data Systems: An Exploratory Survey“. In Software Quality: Quality Intelligence in Software and Systems Engineering, herausgegeben von Dietmar Winkler, Stefan Biffl, Daniel Mendez, und Johannes Bergsmann, 371:13–27. Lecture Notes in Business Information Processing. Cham: Springer International Publishing. <https://doi.org/10.1007/978-3-030-35510-4_2>.
18. Felderer, Michael, Barbara Russo, und Florian Auer. 2019. „On Testing Data-Intensive Software Systems“. arXiv:1903.09413 [cs], April. <http://arxiv.org/abs/1903.09413>.
19. English, L.P. Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1999.
20. Wang, Richard Y., und Diane M. Strong. 1996. „Beyond Accuracy: What Data Quality Means to Data Consumers“. Journal of Management Information Systems 12 (4): 5–33. <https://doi.org/10.1080/07421222.1996.11518099>.
21. Huang, Xiaowei, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, und Xinping Yi. 2020. „A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability“. arXiv:1812.08342 [cs], Mai. [http://arxiv.org/abs/1812.08342](http://arxiv.org/abs/1812.08342" \t "_blank).
22. Kim, Jinhan, Robert Feldt, und Shin Yoo. 2018. „Guiding Deep Learning System Testing using Surprise Adequacy“. arXiv:1808.08444 [cs], August. [http://arxiv.org/abs/1808.08444](http://arxiv.org/abs/1808.08444" \t "_blank).
23. Ma, Lei, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, u. a. 2018. „DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems“. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 120–31. Montpellier France: ACM. <https://doi.org/10.1145/3238147.3238202>.
24. Pei, Kexin, Yinzhi Cao, Junfeng Yang, und Suman Jana. 2017. „DeepXplore: Automated Whitebox Testing of Deep Learning Systems“. In Proceedings of the 26th Symposium on Operating Systems Principles, 1–18. Shanghai China: ACM. <https://doi.org/10.1145/3132747.3132785>.
25. Sun, Youcheng, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, und Rob Ashmore. 2019. „Structural Test Coverage Criteria for Deep Neural Networks“. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 320–21. Montreal, QC, Canada: IEEE. <https://doi.org/10.1109/ICSE-Companion.2019.00134>.
26. Wicker, Matthew, Xiaowei Huang, und Marta Kwiatkowska. 2018. „Feature-Guided Black-Box Safety Testing of Deep Neural Networks“. arXiv:1710.07859 [cs], Februar. [http://arxiv.org/abs/1710.07859](http://arxiv.org/abs/1710.07859" \t "_blank).
27. Cheng, Chih-Hong, Georg Nührenberg, Chung-Hao Huang, Harald Ruess, und Hirotoshi Yasuoka. 2018. „Towards Dependability Metrics for Neural Networks“. arXiv:1806.02338 [cs, stat], Juni. [http://arxiv.org/abs/1806.02338](http://arxiv.org/abs/1806.02338" \t "_blank).
28. Dong, Yizhen, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, und Dai Ting. 2019. „There is Limited Correlation between Coverage and Robustness for Deep Neural Networks“. arXiv:1911.05904 [cs, stat], November. [http://arxiv.org/abs/1911.05904](http://arxiv.org/abs/1911.05904" \t "_blank).
29. Huang, X., Kwiatkowska, M., Wang, S., & Wu, M. (2017, July). Safety verification of deep neural networks. In International Conference on Computer Aided Verification (pp. 3-29). Springer, Cham. (e.g. Huang et al., 2017; Ehlers, 2017; Cheng et al., 2017; Tjeng et al., 2018)
30. Ribeiro, Marco Tulio, Sameer Singh, und Carlos Guestrin. 2016. „‚Why Should I Trust You?‘: Explaining the Predictions of Any Classifier“. arXiv:1602.04938 [cs, stat], August. [http://arxiv.org/abs/1602.04938](http://arxiv.org/abs/1602.04938" \t "_blank).
31. Bansal, Aayush, Ali Farhadi, und Devi Parikh. 2014. „Towards Transparent Systems: Semantic Characterization of Failure Modes“. In Computer Vision – ECCV 2014, herausgegeben von David Fleet, Tomas Pajdla, Bernt Schiele, und Tinne Tuytelaars, 8694:366–81. Lecture Notes in Computer Science. Cham: Springer International Publishing. [https://doi.org/10.1007/978-3-319-10599-4\_24](https://doi.org/10.1007/978-3-319-10599-4_24" \t "_blank).
32. Miller, Tim. 2018. „Explanation in Artificial Intelligence: Insights from the Social
33. Chen, T. Y., S. C. Cheung, und S. M. Yiu. 2020. „Metamorphic Testing: A New Approach for Generating Next Test Cases“. <https://doi.org/10.48550/ARXIV.2002.12543>.
34. Gerrard, P. and Thompson, N. (2002) Risk-based e-business testing, Artech House Publishers.
35. Großmann, Jürgen; Felderer, Michael; Viehmann, Johannes; Schieferdecker, Ina: A taxonomy to assess and tailor risk-based testing in recent testing standards In: IEEE Software, Vol.37 (2020), No.1, pp.40-49
36. Felderer, Michael; Großmann, Jürgen; Schieferdecker, Ina: Recent advances in classifying risk-based testing approaches In: Ruggeri, Fabrizio (Ed.): Analytic Methods in Systems and Software Testing. New York: Wiley-Blackwell, 2018, pp. 1-25
37. Felderer, M. and Ramler, R. (2016) Risk orientation in software testing processes of small and medium enterprises: an exploratory and comparative study. Software Quality Journal, 24 (3), 519–548.
38. Erdogan, G; Li, Y.; Runde, R.; Seehusen, F.; Stølen, K.: Approaches for the combined use of risk analysis and testing: A systematic literature review. In International Journal on Software Tools for Technology Transfer, volume 16, pages 627-642, 2014
39. ETSI EG 203 251: Methods for Testing & Specification; Risk-based Security Assessment and Testing Methodologies, 2016
40. D. Xiao, M. Dianati, W. G. Geiger and R. Woodman, "Review of Graph-Based Hazardous Event Detection Methods for Autonomous Driving Systems," in IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 5, pp. 4697-4715, May 2023, doi: 10.1109/TITS.2023.3240104.
41. Harald Foidl and Michael Felderer. 2019. Risk-based data validation in machine learning-based software systems. In Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE 2019). Association for Computing Machinery, New York, NY, USA, 13–18. <https://doi.org/10.1145/3340482.3342743>
42. C. Gladisch, C. Heinzemann, M. Herrmann and M. Woehrle, "Leveraging combinatorial testing for safety-critical computer vision datasets," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 2020, pp. 1314-1321, doi: 10.1109/CVPRW50498.2020.00170.
43. I. Goodfellow, Y. Bengio and Aaron Courville, “Deep Learning”, MIT Press 2016
44. V. Vapnik “Statistical Learning Theory”, John Wiley & Sons 1998
45. N. Harvey, C. Liaw and A. Mehrabian, "Nearly-tight VC-dimension bounds for piecewise linear neural networks" arXiv:1703.02930
46. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In Proceedings of the International Conference on Learning Representations (ICLR), 2014.
47. R. Balan, M. Singh and D. Zou "Lipschitz Properties for Deep Convolutional Networks" arXiv:1701.05217
48. Aws Albarghouthi, "Introduction to Neural Network Verification", arXiv:2109.10317v2

Annex:  
Change History

| Date | Version | Information about changes |
| --- | --- | --- |
| <Month year> | <#> | <Changes made are listed in this cell> |
|  |  |  |
|  |  |  |
|  |  |  |

# History

|  |  |  |
| --- | --- | --- |
| **Document history** | | |
| <Version> | <Date> | <Milestone> |
| 0.0.1 | 2022-11-30 | Early draft |
| 0.0.2 | 2023-03-06 | Restructuring paragraphs 4 and 5 |
|  |  |  |
|  |  |  |

*Latest changes made on 2022-03-14*