ETSI TR 103 910 V1.1.1 (2025-02)

Methods for Testing and Specification (MTS);
AI Testing;
Test Methodology and Test Specification for
ML-based Systems

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

Contents

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 IPR online database.

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™**, **LTE™** and **5G™** logo 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 Methods for Testing and Specification (MTS).

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 (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 present document covers testing of ML-based systems for the purpose of standardization 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 ML-based systems in standardization contexts.

Introduction

Machine Learning (ML) and especially the application of Neural Networks (NNs) have led to 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 GPT-3, 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, there is a growing need to integrate ML models and NNs into software systems that are developed for critical tasks and operate in critical environments. At this point at the latest, the question arises as to how ML, especially NN as well as their integration into systems can be rigorously tested and quality assured. The present document describes methods and approaches for testing such ML-based applications.

The present document intentionally focuses 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.

In summary, the present document provides an introduction as well as procedural understanding of testing ML-based systems. It presents principles and challenges for testing ML-based systems, quality criteria and test items as well as suitable test methods and their integration into the life cycle of typical ML-based applications, with relevance for industry, regulation and research.

1 Scope

The present document describes test types, test items, quality criteria, and testing methodologies associated with testing ML-based systems, with an emphasis on supervised, unsupervised, and reinforcement learning. The present document outlines how these testing practices can be effectively integrated into the life cycle of typical ML-based systems. The present document applies to all types of organizations involved in any of the lifecycle stages of developing and operating ML-based systems as well as to any other stakeholder roles.

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.1] ETSI TR 104 066 (V1.1.1) (07-2024): "Securing Artificial Intelligence; Security Testing of AI".

[i.2] Wang, Richard Y., and Diane M. Strong (1996): "Beyond Accuracy: What Data Quality Means to Data Consumers", Journal of Management Information Systems 12 (4): pp. 5-33.

[i.3] Zhang, J. M., Harman, M., Ma, L. & Liu, Y: "Machine Learning Testing: Survey, Landscapes and Horizons". arXiv:1906.10742 [cs, stat] (2019).

[i.4] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella: "Testing machine learning based systems: a systematic mapping", Empir Software Eng, Bd. 25, Nr. 6, S. 5193-5254, November 2020. doi: 10.1007/s10664-020-09881-0.

[i.5] M. Pol, T. Koomen, and A. Spillner: "Management and optimisation of the testing process: a practical guide to successful software testing with TPI and TMap", updated ed. Heidelberg: dpunkt-Verl, 2002.

[i.6] Poddey A., Brade T., Stellet J. E. & Branz W: "On the validation of complex systems operating in open contexts", arXiv:1902.10517 [cs], 2019.

[i.7] Gal, Yarin: "Uncertainty in Deep Learning", University of Cambridge, October 13th, 2016 .

[i.8] Madry et.al.: "Adversarial Examples Are Not Bugs, They Are Features", arXiv:1905.02175v4.

[i.9] Sahil Verma and Julia Rubin. 2018: "Fairness definitions explained". In Proceedings of the International Workshop on Software Fairness (FairWare '18). Association for Computing Machinery, New York, NY, USA, 1–7. doi: 10.1145/3194770.3194776.

[i.10] M. Borg: "The AIQ Meta-Testbed: Pragmatically Bridging Academic AI Testing and Industrial Q Needs", arXiv:2009.05260 [cs], September 2020, Zugegriffen: October 13, 2021. [Online].

[i.11] ISO 21448:2022: "Road vehicles — Safety of the intended functionality".

[i.12] ISO 26262:2018: "Road vehicles — Functional safety".

[i.13] ISO/IEC/IEEE 24765™:2017: "Systems and software engineering — Vocabulary", in ISO/IEC/IEEE 24765™:2017I , vol., no., pp.1-541, 28 August 2017, doi: 10.1109/IEEESTD.2017.8016712.

[i.14] Beuth Verlag. (2023): "Managing and understanding artificial intelligence: The Practical Guide for Decision Makers, Developers and Regulators", Beuth Verlag.

[i.15] Shen Y., Song K., Tan X., Li D., Lu W., & Zhuang Y. (2024): "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face". Advances in Neural Information Processing Systems, 36.

[i.16] Islam S., Elmekki H., Elsebai A., Bentahar J., Drawel N., Rjoub G., & Pedrycz W. (2023): "A comprehensive survey on applications of transformers for deep learning tasks", Expert Systems with Applications, 122666.

[i.17] Sterner P., Friemelt B., Goretzko,D., Kraus E., Bühner M., & Pargent F. (2024): "The confidence/significance level implies a certain cost ratio between error 1. type and error 2. type", Diagnostica.

[i.18] Haneke U., Trahasch S., Zimmer M., & Felden C. (2021): "Data science: basics, architectures and applications", dpunkt Verlag.

[i.19] Chicco D., Warrens M. J., & Jurman G. (2021): "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation", Peerj computer science, 7, e623.

[i.20] Willmott C. J., & Matsuura K. (2005): "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance", Climate research, 30(1), pp. 79-82.

[i.21] Mathias S. G., Großmann D., & Sequeira G. J. (August 2019): "A comparison of clustering measures on raw signals of welding production data", In 2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML) (pp. 55-60). IEEE.

[i.22] Prasad M., Jagadeeshwar M., & Shanthi D. (2024): "A Comparative Study Of K-Medoids And Fuzzy K-Means Clustering For The Selection Of Optimal Cloud Service Provider", Educational Administration: Theory and Practice, 30(4), pp. 8045-8051.

[i.23] Mahadevan S. (1996): "Average reward reinforcement learning: Foundations, algorithms, and empirical results", Machine learning, 22(1), pp. 159-195.

[i.24] Yu M., Yang Z., Kolar M., & Wang Z. (2019): "Convergent policy optimization for safe reinforcement learning", Advances in Neural Information Processing Systems, 32.

[i.25] Littman M. L., & Szepesvári C. (July 1996): "A generalized reinforcement-learning model: Convergence and applications", In ICML (Vol. 96, pp. 310-318).

[i.26] Guerraoui R., Gupta N., & Pinot R. (2024): "Robust Machine Learning".

[i.27] Wang M., Yang N., Gunasinghe D. H., & Weng N. (2023): "On the Robustness of ML-Based Network Intrusion Detection Systems: An Adversarial and Distribution Shift Perspective", Computers, 12(10), 209.

[i.28] Brown O., Curtis A., & Goodwin J. (2021): "Principles for evaluation of ai/ml model performance and robustness", arXiv preprint arXiv:2107.02868.

[i.29] Braiek H. B., & Khomh F. (2024): "Machine Learning Robustness: A Primer", arXiv preprint arXiv:2404.00897.

[i.30] Thams N., Oberst M., & Sontag D. (2022): "Evaluating robustness to dataset shift via parametric robustness sets", Advances in Neural Information Processing Systems, 35, pp. 16877-16889.

[i.31] Hort M., Chen Z., Zhang J. M., Harman M., & Sarro F. (2023): "Bias mitigation for machine learning classifiers: A comprehensive survey", ACM Journal on Responsible Computing.

[i.32] Mehrabi N., Morstatter F., Saxena N., Lerman K., & Galstyan A. (2021): "A survey on bias and fairness in machine learning", ACM computing surveys (CSUR), 54(6), pp. 1-35.

[i.33] Kersten H., Reuter J., Schröder K. W., & Wolfenstetter K. D. (2008): "IT security management in accordance with ISO 27001 and IT-Grundschutz", Vieweg.

[i.34] Maurer U., Rüedlinger A., & Tackmann B. (2012): "Confidentiality and integrity: A constructive perspective", In Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings 9 (pp. 209-229). Springer Berlin Heidelberg.

[i.35] Xu L., Jiang C., Wang J., Yuan J., & Ren Y. (2014): "Information security in big data: privacy and data mining", IEEE Access, 2, pp. 1149-1176.

[i.36] Wiyatno R. R., Xu A., Dia O., & De Berker A. (2019): "Adversarial examples in modern machine learning: A review", arXiv preprint arXiv:1911.05268.

[i.37] Brendel W., Rauber J., & Bethge M. (2017): "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models", arXiv preprint arXiv:1712.04248.

[i.38] Hanif H., Nasir M. H. N. M., Ab Razak M. F., Firdaus A., & Anuar N. B. (2021): "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches", Journal of Network and Computer Applications, 179, 103009.

[i.39] Akkiraju et al. (2018): "Characterizing machine learning process: A maturity framework", arXiv:1811.04871 [cs], November 2018.

[i.40] Amershi Saleema, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann (2019): "Software Engineering for Machine Learning: A Case Study", in 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291-300. Montreal, QC, Canada: IEEE. doi: 10.1109/ICSE-SEIP.2019.00042.

[i.41] Studer, Thanh, Drescher, Hanuschkin, Winkler, Peters, and Mueller: "Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology", arXiv:2003.05155 [cs, stat], March 2020.

[i.42] ISO/IEC/IEEE 29119-2™:2021: "Software and systems engineering — Software testing — Part 2: Test processes".

[i.43] ISO/IEC 25059:2023: "Software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model for AI systems".

[i.44] ISO/IEC 25010:2011: "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) —System and software quality models".

[i.45] ISO/IEC/IEEE 29148™:2018: "Systems and software engineering — Life cycle processes — Requirements engineering".

[i.46] T. Y.Chen, S. C. Cheung, and S. M. Yiu (2020): "Metamorphic Testing: A New Approach for Generating Next Test Cases". doi: 10.48550/ARXIV.2002.12543.

[i.47] Gerrard P. and Thompson N. (2002): "Risk-based e-business testing, Artech House Publishers".

[i.48] 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.

[i.49] 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.

[i.50] 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), pp. 519-548.

[i.51] 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, pp. 627-642, 2014.

[i.52] ETSI EG 203 251 (01-2016): "Methods for Testing & Specification; Risk-based Security Assessment and Testing Methodologies".

[i.53] Paul Schwerdtner et.al.: "Risk Assessment for Machine Learning Models", arXiv:2011.04328v1.

[i.54] Oliveira S. R., & Zaïane O. R. (11-2003): "Protecting sensitive knowledge by data sanitization". In 3rd IEEE International conference on data mining, pp. 613-616.

[i.55] Arniban Charkroboty et. al.: "Adversarial Attacks and tnces: A Survey". Xiv:1810.00069v1.

[i.56] 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.

[i.57] G. Bernot, L. Bouaziz, and P. Le Gall: "A theory of probabilistic functional testing", in Proceedings of the 19th international conference on Software engineering - ICSE '97, Boston, Massachusetts, United States: ACM Press, 1997, S. 216-226. doi: 10.1145/253228.253273.

[i.58] Pietrantuono R., Russo S. (2018): "Probabilistic Sampling-Based Testing for Accelerated Reliability Assessment", in: 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). Presented at the 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, Lisbon, pp. 35-46. doi: 10.1109/QRS.2018.00017.

[i.59] Wiesbrock H.-W., Großmann J. (2024): "Outline of an Independent Systematic Blackbox Test for ML-based Systems", arXiv, 2024. doi: 2401.17062.

[i.60] Zhang S., Pan Y., Liu Q., Yan Z., Choo K. K. R., & Wang G. (2024): "Backdoor Attacks and Defenses Targeting Multi-Domain AI Models: A Comprehensive Review". ACM Computing Surveys.

[i.61] 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.

[i.62] 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.

[i.63] W. M. McKeeman: "Differential Testing for Software", Digit. Tech. J., pp. 100-107, 1998.

[i.64] 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.

[i.65] 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.

[i.66] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner: "Detecting Adversarial Samples from Artifacts", ArXiv, 2017.

[i.67] 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.

[i.68] I. J. Goodfellow, J. Shlens, and C. Szegedy: "Explaining and Harnessing Adversarial Examples", CoRR, 2014.

[i.69] Wang, Shengrong, Dongcheng Li, Hui Li, Man Zhao, and W. Eric Wong. (2024): "A Survey on Test Input Selection and Prioritization for Deep Neural Networks", In 2024 10th International Symposium on System Security, Safety, and Reliability (ISSSR), 232-43. doi: 10.1109/ISSSR61934.2024.00035.

[i.70] Dang, Xueqi, Yinghua Li, Mike Papadakis, Jacques Klein, Tegawendé F. Bissyandé, and Yves Le Traon (2024): "Test Input Prioritization for Machine Learning Classifiers", IEEE Transactions on Software Engineering 50 (3): 413-42. doi: 10.1109/TSE.2024.3350019.

[i.71] Mosin, Vasilii, Miroslaw Staron, Darko Durisic, Francisco Gomes de Oliveira Neto, Sushant Kumar Pandey, and Ashok Chaitanya Koppisetty (2022): "Comparing Input Prioritization Techniques for Testing Deep Learning Algorithms", In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 76-83. doi: 10.1109/SEAA56994.2022.00020.

[i.72] Demir, Demet, Aysu Betin Can, and Elif Surer (2023): "Distribution Aware Testing Framework for Deep Neural Networks", IEEE Access 11: 119481-505. doi: 10.1109/ACCESS.2023.3327820.

[i.73] Monika Steidl, Ruth Breu, and Benedikt Hupfauf (2020): "Challenges in Testing Big Data Systems: An Exploratory Survey", In "Software Quality: Quality Intelligence in Software and Systems Engineering", published by Dietmar Winkler, Stefan Biffl, Daniel Mendez, and Johannes Bergsmann, 371:13-27. Lecture Notes in Business Information Processing. Cham: Springer International Publishing. doi: 10.1007/978-3-030-35510-4_2.

[i.74] Michael Felderer, Barbara Russo, and Florian Auer (2019): "On Testing Data-Intensive Software Systems", arXiv:1903.09413 [cs], April. doi: 1903.09413.

[i.75] 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.

[i.76] Wang, Richard Y., and Diane M. Strong (1996): "Beyond Accuracy: What Data Quality Means to Data Consumers", Journal of Management Information Systems 12 (4): 5-33. doi: 10.1080/07421222.1996.11518099.

[i.77] Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016): "'Why Should I Trust You?': Explaining the Predictions of Any Classifier", arXiv:1602.04938 [cs, stat], August. doi: 1602.04938.

[i.78] Bansal, Aayush, Ali Farhadi, and Devi Parikh (2014): "Towards Transparent Systems: Semantic Characterization of Failure Modes", In Computer Vision - ECCV 2014, published by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, 8694:366-81. Lecture Notes in Computer Science. Cham: Springer International Publishing. doi: 10.1007/978-3-319-10599-4_24.

[i.79] Miller Tim (2018): "Explanation in Artificial Intelligence: Insights from the Social".

[i.80] Pei, Kexin, Yinzhi Cao, Junfeng Yang, and 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. doi: 10.1145/3132747.3132785.

[i.81] 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. doi: 10.1145/3238147.3238202.

[i.82] Sun, Youcheng, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and 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. doi: 10.1109/ICSE-Companion.2019.00134.

[i.83] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska (2018): "Feature-Guided Black-Box Safety Testing of Deep Neural Networks", arXiv:1710.07859 [cs], February. doi: 1710.07859.

[i.84] Cheng, Chih-Hong, Georg Nührenberg, Chung-Hao Huang, Harald Ruess, and Hirotoshi Yasuoka (2018): "Towards Dependability Metrics for Neural Networks", arXiv:1806.02338 [cs, stat], June. doi: 1806.02338.

[i.85] Kim, Jinhan, Robert Feldt, and Shin Yoo (2018): "Guiding Deep Learning System Testing using Surprise Adequacy", arXiv:1808.08444 [cs], August. doi: 1808.08444.

[i.86] Huang, Xiaowei, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and 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], May. doi: 1812.08342.

[i.87] Dong, Yizhen, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, and Dai Ting (2019): "There is Limited Correlation between Coverage and Robustness for Deep Neural Networks", arXiv:1911.05904 [cs, stat], November. doi: 1911.05904.

[i.88] Aldahdooh A., Hamidouche W., Fezza S. A., & Déforges O. (2022): "Adversarial example detection for DNN models: A review and experimental comparison". Artificial Intelligence Review, 55(6), 4403-4462.

[i.89] Hugging Face: "AI tasks", Retrieved June 23, 2024.

[i.90] Van der Maaten L., & Hinton G. (2008): "Visualizing data using t-SNE", Journal of machine learning research, 9(11).

[i.91] Anowar F., Sadaoui S., & Selim B. (2021): "Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne)", Computer Science Review, 40, 100378.

[i.92] Platzer A. (2013): "Visualization of SNPs with t-SNE", PloS one, 8(2), e56883.

[i.93] Norvig P. R., & Intelligence S. A. (2002): "A modern approach", Prentice Hall Upper Saddle River, NJ, USA: Rani M., Nayak R., & Vyas OP (2015): "An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage", Knowledge-Based Systems, 90, pp. 33-48.

[i.94] Borgonovo E., & Plischke E. (2016): "Sensitivity analysis: A review of recent advances", European Journal of Operational Research", 248(3), pp. 869-887.

[i.95] Fasano G., & Franceschini A. (1987): "A multidimensional version of the Kolmogorov-Smirnov test", Monthly Notices of the Royal Astronomical Society, 225(1), pp. 155-170.

[i.96] Dunkelau J., & Duong M. K. (2022): "Towards equalised odds as fairness metric in academic performance prediction", arXiv preprint arXiv:2209.14670.

[i.97] Lin C. H., Lu M. C., Yang S. F., & Lee M. Y. (2021): "A Bayesian control chart for monitoring process variance", Applied Sciences, 11(6), 2729.

[i.98] Celis L. E., Keswani V. & Vishnoi N. (November 2020): "Data preprocessing to mitigate bias: A maximum entropy based approach", In International conference on machine learning, pp. 1349-1359. PMLR.

[i.99] Jiang B. (March 2018): "Approximate Bayesian computation with Kullback-Leibler divergence as data discrepancy", In International conference on artificial intelligence and statistics, pp. 1711-1721. PMLR.

[i.100] Monteiro R. P., Bastos-Filho C., Cerrada M., Cabrera D. R., & Sánchez R. V. (2021): "Using the Kullback-Leibler Divergence and Kolmogorov-Smirnov test to select input sizes to the fault diagnosis problem based on a CNN model", Learning and Nonlinear Models, 18(2), pp. 16-26.

[i.101] Vieira S. M., Kaymak U., & Sousa J. M. (July 2010): "Cohen's kappa coefficient as a performance measure for feature selection", In International conference on fuzzy systems, pp. 1-8. IEEE.

[i.102] Lakshminarayan K., Harp S. A., Goldman R. P., & Samad T. (August 1996): "Imputation of Missing Data Using Machine Learning Techniques", In KDD, vol. 96.

[i.103] García S., Ramírez-Gallego S., Luengo J., Benítez J. M., & Herrera F. (2016): "Big data preprocessing: methods and prospects", Big data analytics, 1, 1-22.

[i.104] Kalapanidas E., Avouris N., Craciun M., & Neagu D. (November 2003): "Machine learning algorithms: a study on noise sensitivity", In Proc. 1st Balcan Conference in Informatics, pp. 356-365. sn.

[i.105] Kertanah, K., Nurmayanti, W. P., Aini, S. R., Amrullah, L. M., & Sya'roni, M. (2023): "Comparison of Algorithms K-Means and DBSCAN for Clustering Student Cognitive Learning Outcomes in Physics Subject", Kappa Journal, 7(2), pp. 251-255.

[i.106] Zhu Z., Chen M., Zhu C., & Zhu Y. (2024): "Effective defense strategies in network security using improved double dueling deep Q-network", Computers & Security, 136, 103578.

[i.107] Simon D. (2001): "Kalman filtering", Embedded systems programming, 14(6), pp. 72-79.

[i.108] Joy T. T., Rana S., Gupta S., & Venkatesh S. (December 2016): "Hyperparameter tuning for big data using Bayesian optimisation", In 2016 23rd International Conference on Pattern Recognition (ICPR), pp. 2574-2579. IEEE.

[i.109] Bergstra J., & Bengio Y. (2012): "Random search for hyper-parameter optimization", Journal of machine learning research, 13(2).

[i.110] Duesterwald E., Murthi A., Venkataraman G., Sinn M., & Vijaykeerthy D. (2019): "Exploring the hyperparameter landscape of adversarial robustness", arXiv preprint arXiv:1905.03837.

[i.111] Bonet D., Levin M., Montserrat D. M., & Ioannidis A. G. (2024): "Machine Learning Strategies for Improved Phenotype Prediction in Underrepresented Populations", In Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, vol. 29, p. 404. NIH Public Access.

[i.112] Misra P., & Yadav A. S. (2020): "Improving the classification accuracy using recursive feature elimination with cross-validation", Int. J. Emerg. Technol, 11(3), pp. 659-665.

[i.113] Husain H., Ciosek K., & Tomioka R. (March 2021): "Regularized policies are reward robust", In International Conference on Artificial Intelligence and Statistics, pp. 64-72. PMLR.

[i.114] Sinsomboonthong, S. (2022): "Performance Comparison of New Adjusted Min-Max with Decimal Scaling and Statistical Column Normalization Methods for Artificial Neural Network Classification", International Journal of Mathematics and Mathematical Sciences, 2022(1), 3584406.

[i.115] Osugi T., Kim D., & Scott S. (November 2005): "Balancing exploration and exploitation: A new algorithm for active machine learning", In 5th IEEE International Conference on Data Mining (ICDM'05), pp. 8-pp. IEEE.

[i.116] Johnson D. D., Blumstein D. T., Fowler J. H., & Haselton M. G. (2013): "The evolution of error: Error management, cognitive constraints, and adaptive decision-making biases", Trends in ecology & evolution, 28(8), pp. 474-481.

[i.117] Blum A., & Stangl K. (2019): "Recovering from biased data: Can fairness constraints improve accuracy?", arXiv preprint arXiv:1912.01094.

[i.118] Lynn P. (2019): "The advantage and disadvantage of implicitly stratified sampling", Methods, data, analyses: a journal for quantitative methods and survey methodology (mda), 13(2), pp. 253-266.

[i.119] Beretta L., & Santaniello A. (2016): "Nearest neighbor imputation algorithms: a critical evaluation", BMC medical informatics and decision making, 16, pp. 197-208.

[i.120] Ch'ng C. K., & Mahat N. I. (2020): "Winsorize tree algorithm for handling outlier in classification problem", International Journal of Operational Research, 38(2), pp. 278-293.

[i.121] Cheng K., & Young D. S. (2023): "An approach for specifying trimming and Winsorization Cutoffs", Journal of Agricultural, Biological and Environmental Statistics, 28(2), pp. 299-323.

[i.122] Salem A. M. G. (2022): "Adversarial inference and manipulation of machine learning models".

[i.123] Chivukula A. S., Yang X., Liu B., Liu W., & Zhou W. (2023): "Adversarial Machine Learning: Attack Surfaces, Defence Mechanisms, Learning Theories in Artificial Intelligence", Springer International Publishing.

[i.124] Bayani S. V., Prakash S., & Shanmugam L. (2023): "Data guardianship: Safeguarding compliance in AI/ML cloud ecosystems", Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 2(3), pp. 436-456.

[i.125] Mohana S., Shyamala C., Rani E. S., & Ambika M. (2023): "Preserving sensitive data with deep learning assisted sanitisation process", Journal of Experimental & Theoretical Artificial Intelligence, 35(4), pp. 589-616.

[i.126] Pulido-Gaytan L. B., Tchernykh A., Cortés-Mendoza J. M., Babenko M., & Radchenko G. (September 2020): "A survey on privacy-preserving machine learning with fully homomorphic encryption", In Latin American High Performance Computing Conference, pp. 115-129. Cham: Springer International Publishing.

[i.127] Prokop K., Połap D., Srivastava G., & Lin J. C. W. (2023): "Blockchain-based federated learning with checksums to increase security in internet of things solutions", Journal of Ambient Intelligence and Humanized Computing, 14(5), pp. 4685-4694.

[i.128] Kaissis G., Ziller A., Passerat-Palmbach J., Ryffel T., Usynin D., Trask A., ... & Braren R. (2021): "End-to-end privacy preserving deep learning on multi-institutional medical imaging", Nature Machine Intelligence, 3(6), pp. 473-484.

[i.129] Kuzminykh I., Yevdokymenko M., & Ageyev D. (October 2020): "Analysis of encryption key management systems: strengths, weaknesses, opportunities, threats", In 2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T), pp. 515-520. IEEE.

[i.130] Kuzminykh I., Yevdokymenko M., & Ageyev D. (October 2020): "Analysis of encryption key management systems: strengths, weaknesses, opportunities, threats", In 2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T), pp. 515-520. IEEE.

[i.131] Alappat M. R. (2023): "Multifactor Authentication Using Zero Trust", Rochester Institute of Technology.

[i.132] Board, Qualifications, ISTQB: "Standard glossary of terms used in Software Testing" (2014).

[i.133] C(2023)3215 – Standardisation request M/593: "Commission implementing decision of 22.5.2023 on a standardisation request to the European Committee for Standardisation and the European Committee for Electrotechnical Standardisation in support of Union policy on artificial intelligence". Retrieved on 2024.10.06.

[i.134] Shahapure K. R., & Nicholas C. (October 2020): "Cluster quality analysis using silhouette score". In 2020 IEEE 7th international conference on data science and advanced analytics (DSAA), pp. 747-748. IEEE.

[i.135] Forero P. A., Kekatos V., & Giannakis G. B. (2012): "Robust clustering using outlier-sparsity regularization". IEEE Transactions on Signal Processing, 60(8), pp. 4163-4177.

[i.136] Moore A. W. (2001): "Cross-validation for detecting and preventing overfitting". School of Computer Science Carneigie Mellon University, 133.

[i.137] Chikodili N. B., Abdulmalik M. D., Abisoye O. A., & Bashir S. A. (November 2020): "Outlier detection in multivariate time series data using a fusion of K-medoid, standardized euclidean distance and Z-score". In International Conference on Information and Communication Technology and Applications, pp. 259-271. Cham: Springer International Publishing.

[i.138] Hallin M., Mordant G., & Segers J. (2021): "Multivariate goodness-of-fit tests based on Wasserstein distance".

[i.139] Diaz-Sanchez D., Marín-Lopez A., Mendoza F. A., Cabarcos P. A., & Sherratt R. S. (2019): "TLS/PKI challenges and certificate pinning techniques for IoT and M2M secure communications". IEEE Communications Surveys & Tutorials, 21(4), pp. 3502-3531.

[i.140] Adida B. (April 2008): "Sessionlock: securing web sessions against eavesdropping". In Proceedings of the 17th international conference on World Wide Web, pp. 517-524.

[i.141] Yee C. K., & Zolkipli M. F. (2021): "Review on confidentiality, integrity and availability in information security", Journal of ICT in Education, 8(2), pp. 34-42.

[i.142] Bailey C., Chadwick D. W., & De Lemos R. (December 2011): "Self-adaptive authorization framework for policy based RBAC/ABAC models". In 2011 IEEE 9th International Conference on Dependable, Autonomic and Secure Computing, pp. 37-44. IEEE.

[i.143] Zhou M., Gao X., Wu J., Grundy J., Chen X., Chen C., & Li L. (July 2023): "Modelobfuscator: Obfuscating model information to protect deployed ml-based systems". In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 1005-1017.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

adversarial attack: technique used in machine learning where subtle modifications are made to input data to deliberately mislead a model and cause it to make incorrect predictions or decisions

adversarial example: carefully crafted input which mislead a model to give an incorrect prediction

NOTE: From [i.1].

adversarial impact: intended/unintended exploitation or manipulation of vulnerabilities, leading to unintended, harmful, or deceptive outcomes

adverse circumstances: situations where a model's functionality deteriorates significantly caused by variations in input data distribution, presence of outliers and noise, adversarial attacks, changes in the environment, as well as shifts in the underlying data generating process

compromittation: state or process where a model has been weakened, manipulated, or breached in a way that undermines its behaviour

data preprocessing: process of transforming raw data into a clean and usable format by handling missing values, normalizing data, encoding categorical variables, and other techniques to enhance the performance of machine learning algorithms

decision-making process (ML): sequence of steps in which a model or system evaluates input data, applies learned knowledge or algorithms, and selects the best possible action, e.g. prediction, or classification to achieve a desired outcome that informs choices or actions

NOTE: For instance, in autonomous vehicles, the decision-making process involves selecting actions like braking or steering based on sensor data to navigate safely through traffic. In fraud detection systems, decision-making is used to classify transactions as legitimate or fraudulent based on patterns learned from historical data.

Deep Neural Network (DNN): type of artificial neural network with multiple hidden layers between the input and output layers, allowing it to model complex patterns and relationships in data

defect: flaw in a component or system that can cause the component or system to fail to perform its required function

NOTE: A defect, if encountered during execution, may cause a failure of the component or system [i.132].

error: human action that produces an incorrect result

NOTE: From [i.132].

exploitation: in reinforcement learning, process of an agent leveraging its current knowledge to select actions that maximize immediate or expected rewards based on past experience

exploitation (cybersecurity of ML): use of model vulnerabilities, whether intentional or unintentional, to gain insights, influence predictions, or manipulate the model's behaviour in ways that compromise its quality

exploration: in reinforcement learning, process of an agent trying out different actions to gather information about the environment, aiming to discover strategies that maximize long-term rewards

failure: deviation of the component or system from its expected delivery, service, or result

NOTE: From [i.132].

fault: flaw in a component or system that can cause the component or system to fail to perform its required function

NOTE: See also defect.

hybrid AI system: AI system in which merged methods are implemented

NOTE: In neurosymbolic AI, inference is carried out using a merged combination of symbolic methods and neurons. This differs from AI methods that work side by side and are not merged with each other.

hyperparameter tuning: process of optimizing the parameters of a machine learning model that are set prior to the learning process to improve the model's performance on a given dataset

inference: refers to using a model to analyse new data and generate predictions or identify patterns

NOTE: This distinguishes inference from exploration, which involves experimenting with actions within an environment to discover optimal strategies, and exploitation, which uses known knowledge to optimize outcomes and doesn't process new data in comparison to inference.

ML algorithm: defined procedure or set of rules implemented within an ML model to process data, enabling the ML model to perform tasks associated with ML model methods, such as supervised, unsupervised, and reinforcement learning

ML model: representation created from abstracted knowledge that analyses data using ML algorithms to identify patterns and relationships using ML model capabilities

ML model capability: processing functionality of an ML model that is executed using an ML method and can be used to fulfil a task

ML model method: overall strategy that uses mathematical models and/or formulas and can be implemented as an ML algorithm

model assessment: assessment of an ML model's performance using various metrics and techniques to determine its effectiveness and generalizability on unseen data

model behaviour: way a model responds to various inputs and produces outputs, reflecting its correctness, robustness and avoidance of unwanted bias

model exploration: process by which an agent actively seeks out new information about its environment in order to improve its understanding of that environment

model inference: process of using an ML model gain insights from data and/or make predictions

Neural Network (NN): computational model inspired by the human brain, consisting of interconnected nodes (so called neurons) that process information in layers to recognize patterns and make predictions

reinforcement learning: method of ML where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards

sensitive entity: sensitive attribute depicted by data points that can depict information from different perspectives and be of non-technical as well as technical nature

NOTE: For instance, such attributes can be human-related attributes such as gender, age, ethnicity and name; Personally Identifiable Information (PII) such as addresses and phone numbers; financial data such as income, credit history and debt; or technical information subject to intellectual property rights.

specified conditions: conditions that depict the detailed criteria, requirements, or parameters that should be met by a product, service, process, or system to demonstrate compliance with relevant standardization as well as legislation documents

supervised learning: method of ML where a model is trained on a labelled dataset, meaning it learns to map input data to known output labels, enabling it to make predictions on new, unseen data

test criteria: set of requirements or conditions used to test the quality, performance of a model, system, software, or component

test data set: data set that is used after training to test the generalizability of the ML model where data are selected independently of the training data but should have the same probability distribution as the training data set

test item: work product (e.g. system, software item, requirements document, design specification, user guide) that is an object of testing

training data set: data sets 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: software-based infrastructure that enables an efficient training process by providing software tools to data preparation and training of ML models

training process: process for building an ML model using a specific training infrastructure and a set of input data or scenarios that consists of activities that select and prepare the training input in order to tune the model so that it is able to generalize beyond the training inputs

unsupervised learning: method of ML where the model identifies patterns, structures, or relationships in data, focusing on discovering hidden patterns or organizing data in meaningful ways

NOTE: Most unsupervised learning algorithms do not require training data, but exceptions exist, such as Self-Organizing Maps (SOMs), which use input data to organize and map patterns.

validation datasets: data sets that are used to tune the hyperparameters of a model, in particular, to prevent overfitting of the model to the training data

3.2 Symbols

Void.

3.3 Abbreviations

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

AES	Advanced Encryption Standard
AI	Artificial Intelligence
AMI	Adjusted Mutual Information
API	Application Programming Interface
ARI	Adjusted Rand Index
AUC	Area Under Curve
BLEU	BiLingual Evaluation Understudy
CMM	Capability Maturity Model
CNN	Convolutional Neural Network
DBI	Davies-Bouldin Index
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDoS	Distributed Denial of Service
DevOps	Development and Operations
DNN	Deep Neural Network
DQN	Deep Q-Network
GNN	Graph Neural Network
GPT-3	Generative Pre-trained Transformer 3
GPU	Graphics Processing Unit
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and Communications Technology
ID	IDentifier
IDPS	Intrusion Detection and Prevention System
IQR	InterQuartile Range
ISTQB	International Software Testing Qualifications Board
KNN	K-Nearest Neighbours
KPI	Key Performance Indicator

MAE	Mean Absolute Error
MFA	Multi-Factor Authentication
ML	Machine Learning
MLOps	Machine Learning and Operations
MR	Metamorphic Relation
MSE	Mean Squared Error
MT	Metamorphic Testing
NLP	Natural Language Processing
NMI	Normalized Mutual Information
NN	Neural Network
ODD	Operational Design Domain
PCA	Principal Component Analysis
PII	Personal Identifiable Information
PoC	Proof of Concept
RBAC	Role-Based Access Control
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
ROC	Receiver Operation Characteristics
SBT	Search Based Testing
SL	Supervised Learning
SOTIF	Safety Of The Intended Functionality
SSL	Secure Socket Layer
SUT	System Under Test
SVM	Support Vector Machines
TLS	Transport Layer Security
t-SNE	t-distributed Stochastic Neighbor Embedding
UCB	Upper Confidence Bound
UL	Unsupervised Learning
UMAP	Uniform Manifold Approximation and Projection
VPN	Virtual Private Network

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. Due to science-oriented debates, a few lines are left to the term learning subsequently. The term learning might not be entirely accurate when applied to nowadays ML-based systems, as their processes differ fundamentally from human understanding. While ML models can identify patterns, make predictions, and adjust their responses based on data inputs, they lack the deeper comprehension and awareness that characterize human cognition. ML-based systems do not understand in the way humans do; they manipulate data and produces outcomes based calculations like statistical correlations, without any subjective awareness or experience of the meaning behind the data. Thus, referring to ML's adaptation as learning can be misleading when learning is associated with human intelligence. Nevertheless, the ML algorithm-related term learning has become established in AI research and development, which was also taken into account for the present document.

Examples of ML algorithms are neural networks, regression models, decision trees, Bayesian inference and kernel-based methods.

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. Unlike classical software, where the structure (or code) directly determines how the program functions, the structure of a Machine Learning (ML) model has less impact on the specific tasks the model performs. Instead, the structure influences other important factors, such as:

- **How well the model learns** (its ability to adapt and improve with additional data).
- **Robustness** (how well it performs under negative impacts, e.g. noise in data).
- **Transparency** (clarity regarding the model's decision-making).
- Other non-functional characteristics like **efficiency, scalability, or generalization**.

Looking 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 as classical software.
- 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.
- If ML algorithms are used to find the best possible solution, however, for complex problems, these solutions are not error-free. While many ML-based systems can be designed to behave deterministically (producing the same output given the same input), randomness can be introduced through random initialization, data shuffling, or stochastic optimization methods, making the system's output non-deterministic. Additionally errors and deviations can arise due to the probabilistic nature of many algorithms (predictions are based on likelihood, not certainty), training limitations (insufficient or unrepresentative data) and model complexity (more complex models can be prone to overfitting or learning noise).

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 should become the subject of more intensive testing.

4.2 Classification of ML methods

Typical areas of application for ML methods are real-time decisions, navigation for robots, game playing, and all areas in which the independent acquisition of knowledge and skills is involved [i.2]. Figure 1 classifies the umbrella term ML into the three main subgroups Unsupervised Learning (UL), Supervised Learning (SL) and Reinforcement Learning (RL). For each group are outlined examples for correspondent algorithms and approaches. In the following is given an overview of specificities related to ML methods [i.14], [i.93]:

- **Supervised learning:** Within the landscape of supervised learning, where models are explicitly trained on labelled datasets. Labelled training data serves as a basis for the model, allowing it to discern patterns and relationships between input features and corresponding target labels.
- **Unsupervised learning:** In the domain of unsupervised learning, the distinctive characteristic lies in the absence of labelled examples, challenging the model to autonomously identify inherent patterns, structures, or relationships within the input data. This autonomous pattern discovery forms the core of the conceptual rationale, model building procedure, and deployment process for unsupervised learning models.
- **Reinforcement learning:** In reinforcement learning, models are designed to learn through interaction with an environment, receiving feedback in the form of rewards or penalties. This distinctive learning paradigm focuses on sequential decision-making, where the model aims to make a series of decisions over time to maximize cumulative rewards. The conceptual rationale, model building procedure, and deployment process in reinforcement learning are intricately tied to these fundamental principles.

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. Reinforcement learning generally focusses on interaction with the environment for maximization of specific variables and suits well for optimization approaches [i.14].

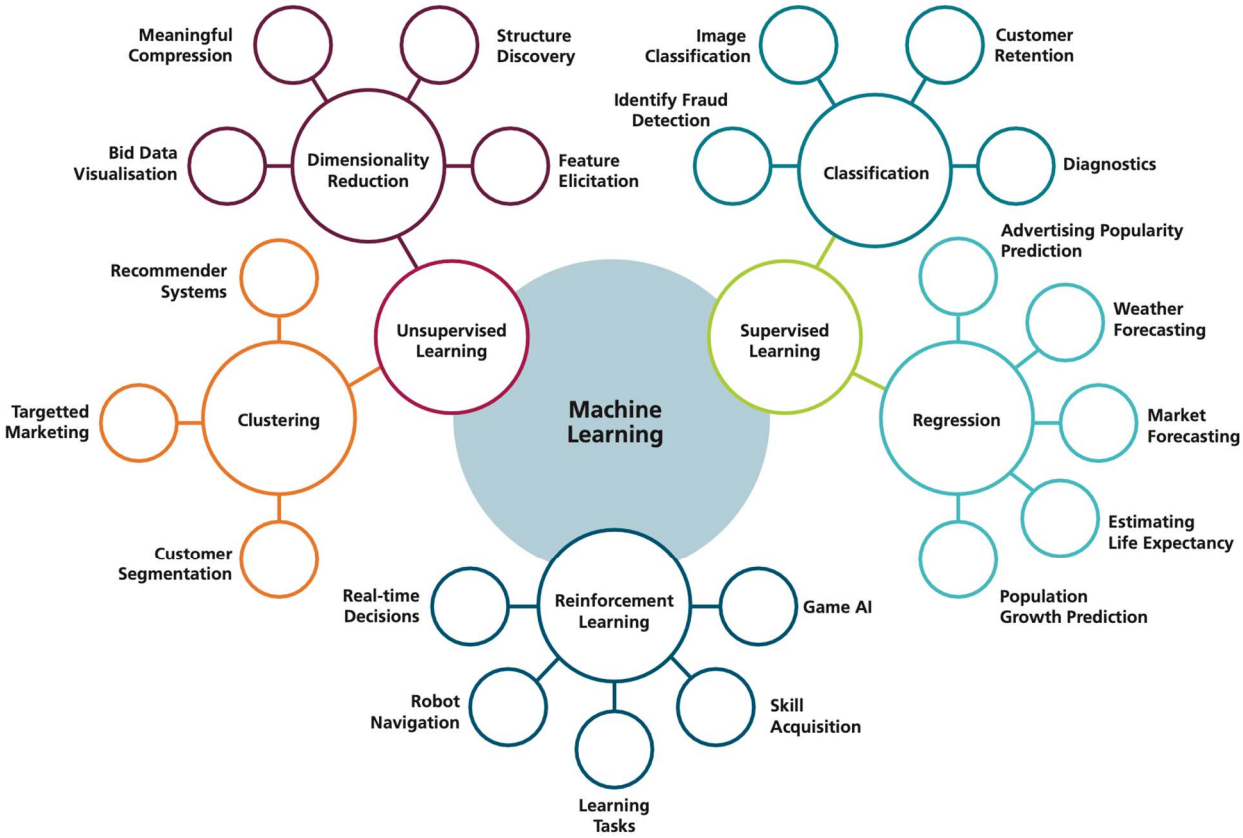

Figure 1: Different ML methods and their fields of application

4.3 ML-based systems and its integration

In the context of quality assurance and testing, ML models cannot be considered in isolation. The effectiveness of an ML model depends on many interconnected factors, including the quality of the data used for training, the surrounding system it interacts with, and the real-world environment it will operate in. A robust testing process should consider not just the model's performance in a controlled environment, but also how it integrates with other systems, handles diverse and evolving data, and responds to real-world variables. In this context, ML models are trained, integrated, and applied within a particular technical and often physical environment. Following this, the technical environment of an ML model and the application environment are distinguished. While influence can usually be exerted 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 a very strong connection between an ML model and its environment, it's essential to test the model alongside the software used for data acquisition, preprocessing, decision postprocessing, and presentation. This is because the model's performance depends on its internal algorithms, the quality of the data it receives, how its outputs are interpreted and how the results or decisions generated by the ML model are displayed or communicated to end users.

Unlike traditional software, the dependencies between an ML model and its surrounding components are often more complex and harder to define than the integration characteristics typically found in classical software systems:

- 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 should 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 may be safeguarded and monitored by dedicated software components to ensure a reliable performance over time.
- Usually, ML models are developed with a focus on its intended use. In addition to the algorithmic nature, in case of training data, any deviations between the data used during training and the data encountered in the actual operational environment can have a significant impact on the model's performance. Therefore, it is essential to subject the training process to thorough quality assurance to ensure the model functions reliably in real-world conditions.

In order to define test specifications at a fine granular level, there are challenges with ML methods in terms of explainability: While ML models based on neural networks are little or not understandable, linear models and decision trees are relatively comprehensible.

Finally, the development of high-quality models requires collaboration from different disciplines. The coordination effort and communication requirements are correspondingly high and should be sufficiently taken into account in the organization of quality assurance.

4.4 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 Zhang et al. define machine learning testing as any process aimed at identifying machine learning defects in machine learning systems [i.3].

On the one hand, this definition shows that testing ML is concerned with 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 should 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, the present document emphasizes 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.

According to Riccio et al. [i.4] it is insufficient for testing methods to only uncover misclassifications and predictive errors within the machine learning model. Instead, the broader implications of such inaccuracies on the entire system should be considered. Isolated misclassifications or incorrect predictions do not adequately define failures, as they might have negligible impact or, conversely, cause the entire system to significantly deviate from its intended functionality and lead to a failure. In addition Pol et al. [i.5] define testing ML-based systems as planning, preparing, and measuring activities designed to assess the characteristics of ML-based systems and to identify the discrepancies between their actual state and their intended state.

In the course of the present document, it will be worked out which test approaches, test items, and principles can usefully be applied to the testing of ML-based systems. It will be investigated 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, it will be addressed 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.

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. [i.6]). Any specification is subject to assumptions that lead to an incomplete or unreliable deduction of the purpose (i.e. what is considered as a useful service to be accomplished by a system), context (i.e. the technical and societal environment of a system) and realization (i.e. the actual implementation) of a system.

This has serious consequences for testing. Given that the context of a system cannot be fully determined, nor, consequently, its purpose, an objective basis for testing the system is lacking. Missing specification means that expressing uncertainty in knowledge during specification places an 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 should 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 should 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 should be accepted. The aim of the optimization process is to reduce this error rate to an acceptable level.

In conventional test processes, a set of test cases is executed and then the subset of failed test cases is analysed. In order to derive a statistical value from this, the relative frequency of failed test runs can be considered, i.e. the number of failed test cases divided by the number of all test cases. Unfortunately, this measure cannot be directly transferred to ML-based systems due to the statistical nature of ML. Instead of the relative frequency, the individual test cases should be weighted by their empirical probability, i.e. the probability of occurrence should be given for each test case, not just for the failed one. Then the total probability for the occurrence of all executed test cases should be calculated and also for the failings. Their quotient gives the correct quality measure for ML-based 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 at 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, which turns a set of numbers into probabilities that add up to 1, helping to decide which class is most likely in multi-class classification tasks), 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 [i.7]. The provision of a reliable confidence value would make it possible to also design safety-critical applications more reliably. If 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 [i.8].

5.4 Need for fair decision making

ML-based systems are increasingly being used to make decisions that can significantly impact people's lives and wellbeing, such as in lending, hiring, criminal justice, and healthcare. Ensuring fair decision making is essential to prevent discrimination, bias, and unfair treatment of individuals or groups based on sensitive attributes like race, gender, or socioeconomic status. Fairness promotes ethical and just decision-making.

Several metrics have been developed to measure and ensure fairness [i.9]. Statistical metrics like statistical parity, predictive parity, equal opportunity etc. focus on balancing outcomes across groups based on observed data. They are easy to compute but may not capture complex fairness issues. Similarity-based metrics emphasize on treating similar individuals similarly. This is requiring a clear definition of similarity by means of e.g. distinct distance metrics. Causal reasoning like counterfactual fairness and causal diagrams aims to understand and adjust for underlying causal relationships, offering a deeper insight into fairness but requires detailed causal knowledge.

Each of these approaches has its strengths and limitations. In practice, the choice of approach depends on the specific context, the available data and the respective fairness requirements. Often, a combination of different metrics can be used to comprehensively consider fairness in machine learning systems. However, the results of the individual fairness metrics often contradict each other, leading to complex trade-offs that need to be considered in practice. Due to the complex constraints in the definition of fairness and the size of the topic, comprehensive coverage of the topic will not be possible, even though it is now known that mitigation of unwanted bias is critical for acceptance and compliance regarding existing EU regulations [i.133]. In the development of ML models, the avoidance of unwanted bias is essential for ensuring fair decision-making. Bias can emerge from various sources, such as the training data, algorithm design, or the operational environment. On the one hand, data collection and preprocessing should ensure that datasets are representative of diverse populations, eliminating any systemic biases. On the other hand, algorithms with bias-related constraints can be implemented to foster equitability-related treatment regarding the intended use. During system operation, continuous conformity assessment is essential to detect any emerging biases regarding technical quality criteria, ethics-related as well as legislative requirements over time. Such continuity in the assessments is important, as the underlying data distribution and the specificities of requirements can change over time.

5.5 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:

- Failure: deviation of the component or system from its expected delivery, service, or result.
- 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.
- 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. [i.3] 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 (e.g. data) as carriers of a fault.

The issue regarding an extended description of ML-related faults is also addressed by Borg et al. by introducing the terms snag and dug. The term "bug" is often too narrowly associated with source code defects and doesn't encompass all functional insufficiencies in ML-based systems. Therefore, the term "snag" is proposed to describe discrepancies between the actual and desired behaviours in ML-based systems, which involve both data and source code. While the root cause of a snag can be a bug in the code or infrastructure, it is frequently due to inadequate training data, a phenomenon termed "dug" by Borg et al. [i.10].

Finally, Humbatova et al. [i.4] 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 should 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 [i.11] is concerned with ensuring the Safety Of The Intended Functionality (SOTIF) in the absence of a failure. It 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. use the term **intended** in the same way it is **explicitly expressed** and as applied in the context of ISO 26262 [i.12] and SOTIF in the terms intended functionality and intended behaviour.

Lastly, it is important to consider whether deviations and errors are inherently part of stochastic, deterministic, or probabilistic methods turned into ML algorithms, by their very nature. This can be traced back to two reasons.

On one hand, Machine Learning (ML) is an optimization process that seeks to approximate a specific objective by adjusting a set of parameters to best fit a given dataset. Separating the data into training, testing, and validation sets helps detect overfitting and allows for the measurement of generalization capabilities. However, the overall optimization process involves trade-offs between different model characteristics, ensuring that, on average, a model performs as expected. This inherently means that there will be situations in which a model's decision does not represent an optimal choice or could even be considered incorrect.

On the other hand, the predictions of an ML-based system can be based on various approaches, including statistical inference, optimization techniques, and pattern recognition methods. While many models leverage statistical principles, particularly in probabilistic frameworks such as Bayesian approaches, others rely on non-statistical methods like neural networks and decision trees. Consequently, uncertainties and deviations can arise from various sources, including the statistical nature of inference processes, the complexities of the models, the quality of the data, and the limitations of the chosen algorithms. This necessitates the use of appropriate criteria, both statistical and non-statistical, to define deviations from expected outcomes. While statistical inference focuses on drawing conclusions about data based on probabilistic models and hypothesis testing, ML algorithms emphasize predictive accuracy and pattern recognition with less reliance on strict assumptions.

Thus, a single counter example may not immediately be considered a violation of the intended purpose. Rather, the failure should 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.

5.6 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.13]. 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 [i.5]. 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.

Regarding ML models, validation and test have a slightly different meaning. In general, in the context of ML models, validation depicts the testing of ML models using validation data and/or specific metrics. This allows the quality of a trained ML model to be assessed before it is set for improvement. This is to be distinguished from validation in system and product development, depicting analytical activities of testing and quality assurance as normally carried out for software systems. Firstly, these analytical activities are much more far-reaching than just testing the basic performance criteria such as overfitting 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 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.

Regarding supervised learning models, validation enables to understand whether the ML model generalizes well to unknown data or whether it has been overtrained on the training data (overfitting). Both validation and evaluation are bound to dedicated data sets. Validation datasets are used before tuning the hyperparameters of a model. In particular, to prevent overfitting of the model to the training data. After tuning, evaluation data sets are used 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 evaluation data sets belong to the training process and thus are intrinsically bound to the training activities.

Usually, validation in unsupervised learning models is distinct from supervised learning due to the absence of labelled data. Without ground truth labels, assessing the quality of model outputs requires different strategies. Internal validation metrics, such as the silhouette score and Davies-Bouldin index, help evaluate the quality of clusters formed by algorithms. The validation process can be iterative, where insights gained from validation can lead to adjustments in model parameters or preprocessing techniques. Building on validation in unsupervised learning, the model's ability to discover meaningful patterns within input data can be improved.

Unlike supervised and unsupervised learning, reinforcement learning involves continuous interaction with an environment, making validation a more complex and ongoing process. Validation can be carried out by analysing metrics/measures regarding cumulative rewards, convergence to optimal policies and balancing exploration with exploitation. Building on validation in reinforcement learning, the model's ability to effectively interact with the environment and learn optimal policies can be enhanced through continuous feedback and performance assessment.

6 Quality criteria addressed by testing ML-based systems

6.1 General

This clause focuses on quality criteria, against which ML models can be tested. While ISO/IEC 25059 [i.43] depicts several quality criteria, the present document lacks for instructions to detect as well as mitigate negative impacts on these criteria. As this shortcoming is a fundamental deficiency of the standardization landscape, this clause addresses the identification of sources for potential negative causes on quality attributes as well as how to detect and avoid these causes. With this background, the following quality attributes were identified to be a prerequisite for assessing the quality of ML-based systems and are described in this clause as well as in the Annexes A, B, C, D and E in detail:

- Model relevance;
- Correctness;
- Robustness;
- Avoidance of unwanted bias;
- Information security;
- Safeguards against exploitation of ML models;
- Explainability.

Among these quality characteristics, model relevance and explainability are identified and tested in fundamentally different ways compared to the other attributes correctness, robustness, avoidance of unwanted bias, information security, and security from vulnerabilities due to their subjective and context-dependent nature. Both model relevance and explainability necessitate the involvement of stakeholders who have developed an understanding of the ML model's mechanisms and can assess its feasibility concerning quality and specific objectives.

Explainability concerns the extent to which a model's mechanisms can be understood by humans, making it different from the other quality criteria. While model relevance refers to how well a model's predictions align with specific goals and stakeholder needs, the criterion can be assessed through judgements on the model's utility in a given context. For testing model relevance, the consideration as well as accessibility of the other quality characteristics is indispensable. In addition, testing model relevance often involves domain-specific metrics and direct input from end-users or stakeholders, making it inherently subjective and context-dependent [i.14].

In principle, test criteria for ML models vary depending on the model's specificity, for instance, due to differences between supervised, unsupervised, and reinforcement learning. This specificity requires adjusting test characteristics according to the specificities of the ML method implemented in an ML model. For instance, in contrast to purely rule-based methods and within the broader machine learning spectrum, supervised machine learning enables the development of ML models based on labelled data. Since the development of a supervised learning model is data-driven, additional testing of training datasets represent an additional degree of freedom and are required to sufficiently assess an ML model, additionally to its algorithmic specificities.

Regarding ML models, common causes can have a negative impacts on the quality criteria correctness, robustness, avoidance of unwanted bias as well security from vulnerabilities, such as poor quality data, biased datasets, or overfitting to specific patterns during training. These issues can lead to incorrect predictions, a model's inability to handle slight variations in input (lack of robustness), or the model's unwanted bias regarding certain groups. With this background, negative impacts on correctness, robustness, and avoidance of unwanted bias can make a model more vulnerable to adversarial impacts. If a model frequently makes incorrect predictions, attackers can exploit these errors to manipulate the model. A lack of robustness makes it easier for adversaries to introduce small input changes that lead to significant misclassifications. Similarly, bias can reveal specific patterns or weaknesses, allowing attackers to target and exploit them.

In addition, in the context of information security, negative causes such as poor data handling, weak encryption, or insecure communication channels can also have a detrimental effect on security from vulnerabilities. For instance, in case of data leakage or breach, or if sensitive data or model parameters are exposed due to inadequate security measures, attackers can gain insights into the model's inner workings and exploit its vulnerabilities more effectively.

In all these cases, the model's vulnerabilities in these areas create easier opportunities for adversarial manipulation can amplify the risk of adversarial impact on ML models. However, when it comes to security from vulnerabilities in general, additional causes come into play, apart from correctness, robustness, avoidance from unwanted bias and information security. Adversarial impacts involve intentional or unintentional inputs designed to manipulate the model's operation, which specifically target its vulnerabilities. These impacts can be protected by safeguarding measures and are unique to security concerns.

With regard to structure, this clause focusses on the description of quality criteria and then refers to four annexes, which can be used as a basis for assessing quality criteria. The appendices are to be understood as guidelines that provide a guide to the explainability of ML-based systems and make it possible to identify negative causes for ML-related quality criteria, measures to identify them and countermeasures to prevent them:

- Annex A: Assessing correctness, robustness, avoidance of unwanted bias of ML models:
  - A.1: Causes related to usual model behaviour;
  - A.2: Causes related to exploiting the model behaviour.
- Annex B: Assessing the information security;
- Annex C: Assessing the safeguards against exploitation of ML model's inference/exploration/exploitation;
- Annex D: Questionnaire for explaining ML-based systems;
- Annex E: ML models: Explaining rationale, development and operation.

6.2 Model relevance

6.2.1 General

Model relevance refers to the extent to which an ML model is applicable for a specific task and encompasses criteria on the following items:

- data for training, inference and exploration;
- ML model methods;
- application context adaptability;
- realizable ML model capabilities;
- suitability for tasks;
- accountability;
- intended use.

Understanding the model relevance enables to assess the algorithm choice, the model's adaptability to different contexts, as well as the deployability of the model under specific conditions.

6.2.2 Criteria for model relevance

Assessing criteria regarding model relevance is essential for demonstrating that machine learning models effectively address the specific needs and requirements of their intended use, Technical key terms such as implemented methods**,** realizable capabilities and used data provide insight into the model's foundation, performance, and adaptability. Additionally, the context-dependent characteristics application context adaptability, accountability, and intended use complement the understanding regarding the model's alignment with user expectations and regulatory requirements.

- **ML model methods:** Depict methods related to supervised learning, unsupervised learning and reinforcement learning. During inference/exploration/exploitation via ML models, the specificity of an ML method impacts the quality of pattern extraction from data. The relevance of a method can be defined by its suitability for a particular task as some algorithms/approaches excel in certain areas while underperform in others [i.14].
- **ML model capabilities:** Depict the ML model's functionality regarding data/information processing and can be realized based on ML model methods. ML model capabilities depict operations with data/information portions and can correspond to functionalities such as [i.14]:
  - **Identification:** Detection of patterns or entities within data/information;
  - **Classification:** Categorizing data/information into predefined groups;
  - **Selection:** Choosing specific subsets of data/information;
  - **Comparison:** Distinguishing between different groups within data/information or identifying how closely these groups correspond or align in characteristics;
  - **Generation:** Creation of data/information based on learned patterns.
- **Intended use:** Refers to the specific purpose or application for which the ML model is developed. It delineates the scope and boundaries within which the model is expected to operate while ensuring specific quality requirements.
- **Application context adaptability:** Adaptability to the application context refers to the ability of a model to function effectively in different contexts. Under changing conditions such as environmental dynamics, such adaptability can be a prerequisite for ensuring quality characteristics during ML model operation. The prerequisite can play a crucial role when models need to adapt to changing circumstances and ensure the correct realization of capabilities.
- **Data:**
  - **Training:** In data-driven ML models, e.g. supervised learning models, model relevance can depend on the quality of training data. Training data of high quality not only enhances the quality criteria of the model but also contributes significantly to its adaptability and relevance in various application contexts;
  - **Inference (SL, UL):** In both supervised learning as well as unsupervised learning, the model's ability to process input data of high quality enables to achieve accurate and meaningful outcomes, e.g. extract useful insights or make reliable predictions. In this context, the model's ability to handle the data's diversity enables to capture a wide range of scenarios, enhancing the model's generalizability across different contexts. Additionally, the processability of the input data's appropriated format and structure is of decisive importance [i.93];
  - **Exploration (RL):** The ML model's ability to handle the diversity in input data can enable to cover a wide range of states and actions across different scenarios, as capturing key features of the environment can influence successful decision-making. Additionally, the model's ability for real-time responsiveness enables a software agent to adapt its strategy dynamically based on new observations, for instance to balance exploration with exploitation. As with inference, the processability of the format and structure of the input data by the ML model is important for exploration [i.93].
- **Accountability:** Establishing clear responsibility within chains of involved parties is essential, e.g. ML-related roles in organizations, including data scientists, developers, and organizations as a whole, along the ML-based system's lifecycle. Such responsibility can be linked to the implication related to the actions of each involved party. The accountability of involved parties can be linked to the specific individuals or organizations responsible for its development and deployment for understanding how outcomes are derived and for identifying who is accountable, e.g. regarding liability, when issues arise [i.14].

- **Suitability for tasks:** Determines how effectively ML models/ML-based systems can address specific problems. In Natural Language Processing, for instance, models should be designed for particular tasks like sentiment analysis or named entity recognition. In contrast, e.g. in Computer Vision, task suitability involves selecting models that excel at image classification, object detection, or segmentation, depending on the application.

6.2.3 Assessing model relevance

6.2.3.1 General

Model relevance in ML is determined by several key factors, including task suitability, capabilities, data quality, and accountability. The algorithm's suitability for the task is crucial, as different models perform better in specific domains, such as classification in Natural Language Processing or object detection in Computer Vision. The model's capabilities-such as selection, recognition, and classification-should align with the task to enhance relevance. High-quality training data significantly influences the model's accuracy and adaptability, especially in supervised learning, enabling the model to perform well in various contexts. The ability to process diverse input data and adapt to changing conditions further strengthens a model's effectiveness across different environments. Finally, accountability across all parties involved in the ML lifecycle, such as data scientists, developers, and organizations, promotes transparent communication among stakeholders, enabling discussions about the strengths and weaknesses of different ML approaches and their implications for specific use cases [i.14].

6.2.3.2 Assessing ML model methods

The assessment of an ML model's methods involves assessing two main aspects: its alignment with the intended use-related quality criteria and its ability to perform specific tasks effectively. The assessment lays the focus on several key points, such as:

- **In case of training: data-driven processing:** Examination of how well the model processes data during the learning phase to ensure it aligns with the intended use and quality criteria. For instance, this can be achieved based on generic measures/metrics like loss function, learning curve and convergence rate.
- **Capability- and task-related measures:** Assessment of the model's algorithm-related performance regarding the execution of specific capabilities and tasks, ensuring it meets the requirements set for its intended use.
- **Rule-based relationships:** Consideration of rule-based relationships, particularly in hybrid AI systems, to understand how rules interact with data-driven components and their impact on the model's overall performance.

6.2.3.3 Assessing ML model capabilities

The assessment of ML model's capabilities enables to understand the realizable functionalities based on the embedded ML model algorithms for an intended use. By analysing the quality of an ML model with different inputs, its compliance with the requirements regarding an intended use case can be determined. For instance, in a Natural Language Processing (NLP) application, the inputs may include text samples from different domains, languages or dialects. The following aspects are relevant to assess the ML model capabilities:

- **Sensitive entity correlations:** Investigation whether the model produces outputs that reflect correlations between sensitive entities.
- **Generic statistical metrics** to assess the operation of ML model capabilities, such as sensitivity, relative error and statistical significance.
- **ML-specific, quality-related measures and metrics** to assess the quality characteristics of a model's outputs, e.g. the BLEU score in NLP or the confidence score.

6.2.3.4 Assessing suitability for tasks

When assessing the relevance of an ML model to achieve specific goals through task execution, it's crucial to discern between ML capabilities and tasks. Tasks describe the processing of data based on the application of the ML model capabilities. In general, the popularity of tasks is generally driven by industrial business models related to AI. According to the current state of the art, tasks of ML include image processing, natural language processing, audio processing deals as well as tabular data processing. As the popularity of ML models able to process various data types increases, multimodal processing is gaining in importance. In this context, a good overview of ML tasks is given at the website of HuggingFace [i.89].

The fulfilment of tasks involves employing methods to realize specific capabilities aimed at achieving desired objectives. Therefore, evaluating the model's suitability, quality characteristics, and limitations for task realization relies heavily on understanding its underlying methods and capabilities. For example, executing the tasks "text summarization" by an ML model involves the utilization of specific ML capabilities alongside ML methods enabling the processing of text. For this, executing the text summarization task involves the use of capabilities such selection of text and keywords, recognition of context, recognition of patterns and user intent. These capabilities can be used throughout the task execution process, both sequentially as well as simultaneously. The execution process itself depends on the software-related programming, defining how algorithms of an in an ML-based system are used.

The concept of model relevance, when described in terms of method and capability pairs, provides a more structured approach to evaluating how well an ML-based system can perform specific tasks. This pairing refers to linking ML methods (such as supervised learning, unsupervised learning, or reinforcement learning) with the model's capabilities (like classification, recognition, or generation). Focusing on these pairings provides a clearer understanding of how the technical components of an ML-based system interact to achieve the desired outcomes [i.15], [i.16]:

- **Granular analysis:** Describing the task execution on the basis of method and capability pairs allows for a more granular analysis of the model's performance. By examining each pair separately, involved parties can gain deeper insights into the model's strengths and weaknesses.
- **Quality assessment:** Understanding how well different methods and capabilities contribute to task execution helps in assessing the quality of the model. It enables involved parties to evaluate whether the model is proficient in utilizing the necessary methods and possesses the required capabilities to accomplish the task effectively.
- **Identification of limitations:** By scrutinizing method and capability pairs, involved parties can identify any limitations or deficiencies in the model. This information is crucial for mitigating risks and making informed decisions about the model's suitability for specific tasks.
- **Optimization opportunities**: Analysing method and capability pairs can clarify opportunities for optimization. Involved parties can identify areas where enhancements or adjustments are needed to improve the model's performance and efficiency in executing tasks.
- **Comparative analysis:** Comparative analysis of different models becomes more straightforward when evaluating method and capability pairs. Involved parties can compare how various models utilize different method and capability pairs to accomplish the same task, facilitating better decision-making.
- **Task-specific insights:** Describing task execution through method and capability pairs provides task-specific insights. Involved parties can understand which methods and capabilities are most crucial for a particular task, guiding the development and selection of appropriate models.
- **Transparency and explainability:** Understanding methods and capabilities behind ML models enhances the transparency and explainability of the model as well as the whole system. Involved parties can clearly see how the model/system operates by examining its underlying methods and capabilities, fostering trust and confidence in its performance.
- **Adaptability:** Having a clear understanding of method and capability pairs allows for easier adaptation of the model to different tasks or environments. Involved parties can identify how the model can be modified or extended to address new challenges or requirements effectively.

6.2.3.5 Assessing application context adaptability

Application context adaptability is fundamental for model relevance due to its role in ensuring that the ML model remains effective across diverse scenarios. In general, to assess the model's adaptability to different application contexts, sensitive entities in input data can be varied to assess corresponding changes in the ML model's output. By assessing the model's adaptability to different application contexts, where sensitive entities in the input data are varied, the parties involved gain insights into the following two dimensions:

1) How well the model adjusts to different contexts in terms of the intended use. Models that can seamlessly adapt to varying circumstances align more closely with the specific needs and objectives of their intended applications, enhancing their overall relevance.

2) How well the model aligns with requirements from legislation and standardization. Different application contexts often come with distinct regulatory frameworks and industry standards, ensuring that the model can adapt while still adhering to these requirements is vital for maintaining compliance and relevance.

6.2.3.6 Assessing accountability

By clarifying the chain of responsibility and liability associated with the development of the model, involved parties can foster the ethical and lawful use of ML-based systems. Assessing accountability for ML models depends on a variety of criteria, i.e. including explicit legal requirements and standardized controls. These assessments require a clear understanding of the obligations of all stakeholders as well as a comprehensive understanding of the causal links between the actions taken and the resulting impact of AI. Such assessments can be conducted based on the following criteria:

- **Legislative compliance:** Compliance with established legal frameworks forms the cornerstone of accountability assessments and ensures that ML models comply with applicable legislative documents.
- **Correspondence with standardization documents:** Reference to standardized guidance and requirements provides a structured framework and enables to justify the choice of specific frameworks and metrics.
- **Clarity of obligations within causal links:** A clear outline of responsibilities between involved parties fosters the understanding of cause-effect-links.

6.3 Correctness

6.3.1 Criteria for correctness

Correctness depicts how closely inference/exploration/exploitation results align with expected outcomes and can be reflected by metrics. These metrics can provide insights into how well a model performs in different tasks, e.g. regarding classification problems in supervised learning, clustering in unsupervised learning and reward optimization in reinforcement learning. Considering a range of metrics tailored to each learning paradigm ensures a more comprehensive assessment of model performance, extending beyond the limitations of specific metrics like accuracy alone [i.17], [i.18].

In the following are given criteria for supervised learning classification:

- **Accuracy** depicts overall correctness by assessing the proportion of correct predictions in all predictions. It provides a high-level overview of a model's performance but may be misleading in cases of imbalanced datasets.
- **Precision** reflects how well a model recognizes positives while avoiding the misclassification of negatives as positives. It quantifies the proportion of true positive predictions among all positive predictions, emphasizing the minimization of false positives. Thereby, precision can be of decisive importance if false positives are costly.
- **Recall** or **sensitivity** (true positive rate) emphasizes the success in finding positives, while its value is related to how many positives are ignored (false negatives). The metric reflects the proportion of true positive predictions among all actual positives. High recall reduces the risk of missing important cases, making it important in scenarios where false negatives are costly.

- **F1 Score** combines precision and recall into a single metric, providing a balanced measure of correctness. It considers both false positives and false negatives, making it suitable for imbalanced datasets where a trade-off exists between precision and recall.
- **Specificity** (true negative rate) quantifies the model's ability to accurately identify instances that do not belong to the positive class. A high specificity indicates that the model is effective at avoiding false alarms and accurately identifying non-events as negative. In comparison to the other measures mentioned among correctness, the specificity is commonly used in the context of binary classification tasks, where there are two possible classes for a specific entity.
- **Type I error** (false positive rate) occurs when the model incorrectly predicts a specific class.
- **Type II error** (false negative rate) occurs if the model doesn't predict a specific class.

The following list summarizes the criteria for **supervised learning regression** [i.19], [i.20]:

- **Mean Absolute Error (MAE):** Measures the average magnitude of errors in a set of predictions, without considering their direction.
- **Mean Squared Error (MSE):** Measures the average of the squares of the errors. It gives more weight to larger errors.
- **Root Mean Squared Error (RMSE):** The square root of the MSE, which brings the error to the same units as the output variable.
- **R-squared (R^2):** Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.
- **Adjusted R-squared:** Adjusts the R^2 value based on the number of predictors in the model.

Below are listed correctness criteria, related to **unsupervised learning** [i.21], [i.22]:

- **Silhouette Score:** Measures how similar an object is to its own cluster compared to other clusters. Values range from -1 to 1, with higher values indicating better-defined clusters.
- **Davies-Bouldin Index (DBI):** Measures the average similarity ratio of each cluster with its most similar cluster. Lower values indicate better clustering.
- **Adjusted Rand Index (ARI):** Compares the similarity between the predicted clusters and a ground truth clustering. Adjusts for chance clustering.
- **Normalized Mutual Information (NMI):** Measures the amount of information shared between the predicted clusters and the true clusters. Values range from 0 to 1, with higher values indicating better clustering.

In the following are correctness criteria for **reinforcement learning** [i.23], [i.24], [i.25]:

- **Cumulative reward:** Sum of all rewards obtained by the agent over an episode or a series of episodes.
- **Average reward:** Mean reward per episode or per time step, providing an indication of long-term performance.
- **Policy convergence:** Stabilization of policy over time, meaning an agent's actions become consistent as it learns.
- **Expected value function convergence:** Convergence of the estimated value functions (state or action values) to their true values.

6.3.2 Assessing correctness

Assessing the correctness of ML models depends on the type of learning - supervised, unsupervised, or reinforcement learning - and typically involves extracting key performance indicators.

In supervised learning, many models output a confidence score alongside their predictions, representing the probability or certainty of the model's prediction. From this, essential metrics such as precision, recall, F1 score, and confusion matrices can be computed to evaluate classification tasks. The confusion matrix helps identify error types like false positives and false negatives, offering insights into model strengths and weaknesses. For binary classification, Receiver Operating Characteristic curves and Area Under the Curve scores assess how well the model balances true positive and false positive rates, complementing precision-recall trade-offs. For regression tasks, metrics like Mean Squared Error or Mean Absolute Error evaluate prediction accuracy. Additionally, cross-validation techniques such as k-fold cross-validation ensure models generalize well to new data.

In unsupervised learning, while a direct confidence score is not typically available, certain models can output metrics that serve a similar purpose. For instance, clustering algorithms may produce measures such as distance from cluster centroids or cluster tightness, which can be interpreted as a form of confidence. In anomaly detection, models often assign a score to each point, indicating how well it fits normal patterns versus being an outlier, which provides insight into how confident the model is in its classifications. Assessing these models involves metrics like Silhouette Score or Davies-Bouldin Index for clustering quality, or using intrinsic measures to assess the model's fit in the absence of labels.

For reinforcement learning, traditional confidence scores don't exist as in supervised learning, but similar concepts emerge through the model's interaction with the environment. In Q-learning or similar methods, Q-values indicate the expected cumulative reward from a given state-action pair, effectively reflecting how "confident" the agent is in the value of its choices. In policy-based methods, policy entropy can measure the agent's uncertainty-lower entropy indicates higher confidence in a specific action, while higher entropy shows uncertainty and exploration. Some advanced RL approaches, like Bayesian reinforcement learning, explicitly model uncertainty, allowing the agent to explore and learn in areas where it is less "confident." Metrics such as average reward per episode, convergence speed, and cumulative reward offer meaningful ways to evaluate RL models by tracking the agent's learning progress and ability to optimize its actions over time.

Annex A provides a framework to assess the characteristics correctness by discussing causes metric and countermeasures with the objective of identifying and preventing negatively influencing characteristics of ML models.

6.4 Robustness

6.4.1 Criteria for robustness

Robustness describes the property of an ML-based system to uphold its functionality in the event of adverse circumstances. Regarding ML-based systems, robustness refers to maintaining functionality in the face of potentially adverse circumstances, such as noisy input data, changes in the environment, or deliberate attacks. Robustness can be used to describe the resilience of an ML-based system if a negative impact on the ML-based system/ML model is not avoided. A negative impact stems from an adverse influence on an ML-based system/ML model, e.g. if security from exploiting did not avoid that interaction. The following aspects depict different perspectives for depicting robustness of ML-based system/ML model [i.26], [i.27], [i.28], [i.29], [i.30]:

- **Operational stability** depicts a model's resilience/low sensitivity against variations in training data as well as while inferring/exploring data. In this context, a robust model maintains consistent operational characteristics, e.g. correctness, across variations. The following items depict different ways needed to reveal robustness.
- **Robustness against shift** refers to the ability of a model to maintain its operational stability if the underlying data distribution changes.
- **Robustness in bias handling** describes an ML model's/system's ability to mitigate negative, bias-related impact.
- **Generalizability** depicts a model's ability to perform well on unseen or new data with no significant drop in operational stability.
- **Sensitivity to outliers and missing data** depicts a model's susceptibility as well as resilience to outliers, noise as well as missing values while processing data.
- **Robustness against adversarial impacts** depicts an ML model's/system's operational stability against adversarial inputs or manipulations, for instance if an adversarial attack is not blocked before interacting with the ML model/system.

- **Robustness in hyperparameter optimization:** Correct identification of optimal or near-optimal hyperparameter settings that consistently lead to good model performance, despite variations in data, model architecture, or hyperparameter search methods.

6.4.2 Assessing robustness

To assess robustness, the processes from the previous clause on correctness can also be repeated for different negative influences. In this case, robustness can be justified, for example, by the fact that correctness is achieved within a tolerance range despite negative influences on the ML model. In this context, for example, the confidence score can be analysed before and during the negative influences. For this, methods such as data augmentation and simulation can be employed. By introducing synthetic data or simulating real-world conditions with noise, outliers, and missing values the model's resilience to variations in input data can be tested. Furthermore, adversarial testing involves generating adversarial examples to evaluate the model's ability to withstand deliberate manipulations or attacks aimed at misleading its predictions. Additionally, by varying hyperparameters, the sensitivity, depicting the possible impact of inconsistent hyperparameters on the ML model's operation can be understood.

In the following is described the assessment methodology on the basis of the robustness criteria:

- **Robustness against shift:**
  - **Domain adaptation tests:** Train the model on one dataset and evaluate its performance on a different, but related, dataset to see how well it adapts to shifts in data distribution.
  - **Simulated shifts:** Introduce artificial changes in the data distribution (e.g. adding noise, changing feature distributions) and observe the impact on model performance.
  - **Performance metrics:** Track key metrics before and after introducing distribution shifts to assess stability.
- **Robustness in bias handling**
  - **Bias auditing:** Analyse model predictions across different demographic groups to detect and quantify bias. Tools like Fairness Indicators can help assess bias in classification tasks.
  - **Adversarial debiasing:** Implement methods to explicitly mitigate bias (e.g. reweighting training examples or using adversarial training) and evaluate their effectiveness.
  - **Performance metrics:** Measure fairness-related metrics, such as disparate impact or equal opportunity, to evaluate how well the model handles bias.
- **Generalizability:**
  - **Train/test split:** Evaluate performance on a separate test dataset that was not used during training to measure generalization capabilities.
  - **Out-of-sample testing:** Test the model on completely new data collected from different sources or under different conditions.
  - **Performance metrics:** Monitor metrics like accuracy and F1 score to ensure that they remain high on unseen data.
- **Sensitivity to outliers and missing data:**
  - **Outlier analysis:** Introduce outliers into the training and test data, and measure the impact on model performance metrics (e.g. accuracy, precision).
  - **Imputation techniques:** Test the model's performance with various methods of handling missing data (e.g. mean imputation, median imputation, predictive modelling).
  - **Performance metrics:** Compare performance metrics across datasets with and without outliers or missing values to assess robustness.

- **Robustness against adversarial impact:**
  - **Adversarial testing:** Generate adversarial examples and assess the model's performance on these examples.
  - **Defense mechanisms:** Implement and evaluate defense strategies, such as adversarial training, e.g. to improve resilience against attacks.
  - **Performance metrics:** Measure changes in performance (e.g. accuracy, loss) when the model is exposed to adversarial examples compared to clean data.
- **Robustness in hyperparameter optimization:**
  - **Reproducibility of results:** Ensure that the optimization process can be reproduced consistently. This allows for reliable comparisons of hyperparameter settings across different runs.
  - **Convergence behaviour:** Analyse the convergence behaviour of the optimization algorithm (e.g. grid search, random search, Bayesian optimization) by monitoring performance metrics over iterations. Robust optimization processes should demonstrate stable convergence patterns.

Annex A offers a framework for assessing the robustness of the characteristics mentioned above.

6.5 Avoidance of unwanted bias

6.5.1 Criteria for avoidance of unwanted bias

Avoidance of unwanted bias is aimed at the assessment of imbalances regarding ML model inference as well as in training data. Unwanted bias in the context of machine learning refers to the presence of unintended disparities, prejudices, or imbalances in the data as well as algorithmic model characteristics. Unwanted bias can lead to imbalanced outcomes, exacerbating inequalities of sensitive entities. Efforts to mitigate unwanted bias involve careful selection and preprocessing of training data, implementing algorithms, carrying out unbiased assessment and techniques designed to reduce or eliminate bias in model predictions. Key criteria to avoid unwanted bias include [i.31], [i.32]:

- **High-quality training data** (e.g. diverse, representative, consistent, correctly annotated): Unwanted bias can be introduced during the model training phase, where the algorithm learns patterns and relationships from historical data. If the training data contains biases, the model may learn and perpetuate those biases. If certain sensitive entities are underrepresented in the training data, the model may exhibit poorer performance for sensitive entities or produce biased predictions. Biased models may contribute to unequal treatment, e.g. discrimination of sensitive entities.
- **Avoidance of biased model exploration/inference:** Avoiding unwanted bias in ML-based systems involves ensuring equitable decision-making processes during both exploration and inference. Avoidance of biased exploration occurs when certain actions or regions of the solution space are not favoured due to inherent biases in the exploration strategy. Avoidance of biased inference occurs if the algorithm's objective function or decision boundaries are not skewed towards certain outcomes, irrespective of inferred/explored data.

6.5.2 Assessing avoidance of unwanted bias

Before assessing if unwanted bias has been avoided, at the beginning, sensitive attributes have to be identified. Using statistics can be understood, how these sensitive attributes are distributed in data. The statistical results can uncover underlying patterns or imbalances that could skew model predictions. For instance, using correctness metrics enables to quantitatively measure bias across various segments of the dataset.

Table 1 gives an overview of procedures describing how to assess the avoidance of unwanted bias by focussing on quality of training data and model exploration/inference:

Table 1: Assessment criteria for avoidance of unwanted bias.

Item	Assessment criteria	Procedures for assessment
Quality of training data	Diversity and representation	• Analyse demographic representation of training data. • Visualize class distributions.
	Consistency	• Check for consistency in data collection methods. • Perform checks for data integrity across sources.
	Correct annotation	• Review labelling processes for accuracy. • Perform audits on a sample of annotated data.
Biased model exploration/inference	Equitable decision-making process	• Evaluate the model's decision-making processes for bias. • Conduct audits on decision outcomes across groups.
	Exploration strategies	• Analyse solution space exploration methods for inherent biases. • Review paths taken during training.
	Unbiased objective functions	• Assess objective function design for potential bias. • Evaluate model performance across different groups.

In this context, Annex A provides a framework to assess the characteristics of unwanted bias by discussing causes metric and countermeasures with the objective of identifying and preventing negatively influencing characteristics of ML models.

6.6 Information security

6.6.1 Criteria for information security

Information security in the context of machine learning focuses on safeguarding data, models, and systems from unauthorized access, tampering, or theft. According to the current state of the art, information security involves protecting sensitive information by insuring confidentiality, integrity and availability [i.33]. For instance, model confidentiality ensures that proprietary algorithms and model parameters are secure to prevent intellectual property theft. Maintaining the integrity of data and models involves restricting access and implementing authentication controls to prevent unauthorized modifications [i.34]. The following ICT-related security criteria are significant to determine the information security of ML models, leaning on [i.35], [i.128], [i.130], [i.141]:

- **Data privacy:** Sensitive data used to train and test ML models is handled and stored with strict privacy controls.
- **Model confidentiality:** Safeguarding and documentation of model parameters and architecture to prevent information leaks and protect intellectual property and proprietary algorithms.
- **Integrity of data and model** restrict unauthorized access to data and ML model.
- **Authentication and access controls:** Implementing strong authentication and access controls ensures that only authorized users or systems can interact with the ML model, reducing the risk of malicious actions that could impact availability.
- **Encryption:** Using encryption to protect data both in transit and at rest is essential. This ensures that even if unauthorized access occurs, the data remains unreadable without the appropriate decryption keys.
- **Secure deployment environment:** Avoidance of negative environmental impacts by implementing firewalls and other network security measures.
- **Physical security:** Protecting physical access to information technology assets, such as servers, data centres, and network infrastructure, is crucial to prevent unauthorized tampering or theft.

6.6.2 Assessing information security

To assess information security of ML models, a structured and detailed approach is essential regarding the criteria mentioned previously. The assessment is described by the following items [i.128], [i.129], [i.130], [i.131], [i.132], [i.141], [i.142]:

- **Data privacy:** Testing for data privacy vulnerabilities involves conducting regular audits of data storage and handling practices to ensure compliance with privacy regulations and policies. These audits **can** look for any signs of unauthorized access or mishandling of sensitive data. Reviewing access logs and permissions periodically is also crucial to ensure that only authorized personnel can access sensitive data. Penetration testing, which simulates attacks on data storage and processing systems, helps identify weaknesses in data protection measures. These tests provide insights into potential vulnerabilities that could be exploited to access sensitive data improperly.

- **Model confidentiality:** Ensuring model confidentiality requires thorough testing of access control mechanisms. Regular reviews and tests of these mechanisms help ensure that only authorized personnel have access to sensitive model details. Additionally, penetration tests can be conducted to simulate attacks on model storage systems to identify potential information leakage points. By monitoring and logging access attempts, it is possible to detect and respond to unauthorized efforts to access or exfiltrate model information.

- **Integrity of data and model:** Testing for vulnerabilities in data and model integrity involves rigorous access control evaluations. Ensuring that authentication methods are secure and that detailed access logs are maintained helps in identifying unauthorized access attempts. Regular security assessments can be conducted to pinpoint and mitigate potential vulnerabilities in data and model storage and processing environments. This includes testing the robustness of the storage solutions and ensuring that data and models are not susceptible to unauthorized modifications.

- **Authentication and access controls:** To test the effectiveness of authentication and access controls, it is essential to conduct regular reviews and tests of the implemented security measures. Multi-Factor Authentication (MFA) and Role-Based Access Controls (RBAC) should be evaluated for their effectiveness in restricting access to the ML models. Regular penetration tests and security assessments help identify weaknesses in the authentication process that could be exploited by malicious actors. These tests ensure that only authorized users or systems can interact with the ML models, reducing the risk of unauthorized access.

- **Encryption:** Testing for encryption vulnerabilities involves conducting audits of the encryption protocols used for data at rest and in transit. Ensuring that industry-standard encryption protocols, such as the Advanced Encryption Standard (AES) for data at rest and Transport Layer Security (TLS)-related controls for data in transit, are in place and properly implemented is crucial. Penetration testing can simulate attempts to intercept encrypted data, helping to identify weaknesses in the encryption process. Additionally, testing key management practices ensures that encryption keys are securely handled and protected.

- **Secure deployment environment:** To test the security of the deployment environment, regular security audits and risk assessments are necessary. These should focus on identifying vulnerabilities in the deployment infrastructure, such as misconfigurations or outdated software. Implementing firewalls and other network security measures, and regularly testing their effectiveness, helps protect against external threats. Intrusion Detection and Prevention Systems (IDPSs) should be tested to ensure they can effectively identify and mitigate potential attacks.

- **Physical security:** Testing for physical security vulnerabilities involves assessing the controls in place to protect access to information technology assets, such as servers, data centres, and network infrastructure. This includes evaluating the effectiveness of access badges, surveillance systems, and security personnel in preventing unauthorized physical access. Regular reviews and updates of physical security measures are necessary to ensure they remain effective against evolving threats.

Regarding the previously mentioned criteria for information security, an overview of potentially negative impacts on information security as well as actions that make these impacts controllable was created. For this, Annex B describes:

- causes that negatively influence information security of ML models for supervised, unsupervised and reinforcement learning components of ML-based systems;

- metrics to detect causes that negatively influence information security of ML models, separated for supervised, unsupervised and reinforcement learning components of ML-based systems;

- countermeasures to mitigate or prevent causes negatively influence information security of ML models, separated for supervised, unsupervised and reinforcement learning components of ML-based systems.

6.7 Safeguards against exploitation of ML models

6.7.1 General

Intentionally as well as unintentional inputs can exploit a model's vulnerabilities, aiming for compromittation of the ML model's behaviour. Adversarial attacks, data poisoning, and even shifts in the data distribution due to regular usage can all lead to unexpected or harmful model behaviour. These exploits can alter predictions, disrupt functionality, or even reveal sensitive information, posing significant risks to both the model's integrity and the broader system it supports. With this background, safeguarding against exploits reflects a distinct category of security measures that can avoid the compromittation of an ML model. Countermeasures like robust input validation, anomaly detection, and regular performance monitoring are essential to preventing these risks (see Annex C). These safeguards ensure that the model's compromittation can be avoided, contribution to a secure intended operation of the ML model.

6.7.2 Criteria for safeguards against exploitation

Ensuring the safeguards of ML models requires addressing multiple layers of potential vulnerabilities. Data validation and sanitization are crucial for filtering out malicious or erroneous inputs, preventing harmful data from reaching the model. Effective resource management strategies, such as rate limiting and load balancing, ensure the model remains stable and responsive under varying conditions. Continuous anomaly detection and monitoring help identify and respond to abnormal patterns or potential attacks, such as adversarial inputs or backdoor injections. Additionally, safeguarding the model's environment from external factors like temperature or humidity, along with obfuscating output data, helps protect against exploitation and preserve the model's integrity.

- **Input data validation:** Ensuring that inputs to the model are secure, correctly formatted, and free from malicious or erroneous data. This involves comprehensive validation, sanitization, and preprocessing to prevent any harmful input from reaching the model.

- **Resource management:** Protecting the model from overload, what includes applying rate limiting, load balancing, and resource management strategies to ensure the system remains responsive and stable under various conditions.

- **Anomaly detection and monitoring:** Continuously tracking the model's behaviour and performance to quickly detect and respond to any abnormal patterns or failures. Real-time monitoring, logging, and alert systems help identify potential exploits or failures before they cause significant damage. This includes detecting and mitigating adversarial inputs, backdoor attacks, and data poisoning, as well as maintaining consistency in model outputs to avoid exploitation.

- **Environment security:** Safeguarding the model's inference and deployment environments from negative impacts, ensuring proper isolation from vulnerabilities. This includes protecting the model from external environmental factors, such as extreme temperatures, humidity, or physical wear, that could affect the hardware or software performance. Exploits can also be advantaged by such environmental conditions.

- **Data sanitization during preprocessing:** Ensuring that all data inputs are thoroughly cleaned and processed before being fed into the model to remove any harmful, malformed, or malicious elements. This includes filtering out noise, handling missing or incomplete data, and removing outliers to prevent skewed model behaviour or adversarial attacks.

- **Obfuscation of output data:** Protecting sensitive or proprietary information from being exposed through the model's outputs. This involves applying techniques such as sanitization, rounding, or noise addition to prevent attackers from reverse-engineering the model's decision-making process or extracting confidential data.

6.7.3 Assessing safeguards against exploitation

- **Validation of input protection:** Assess whether robust input validation techniques are in place to reject malicious or erroneous data before it reaches the model. This can be done by reviewing the model's ability to detect malformed, out-of-range, or adversarial inputs. Testing can include feeding the model with a range of input scenarios, both valid and invalid to verify whether it appropriately rejects harmful data or handles it without compromise.

- **Resource management evaluation:** Assess how the safeguards handle high volumes of data or requests under stress. This involves testing for rate-limiting, load balancing, and system resilience under various conditions, such as heavy traffic or resource exhaustion. By simulating different load scenarios, one can determine if the safeguards ensure the model to remain stable and responsive, without being overwhelmed by excessive requests or degraded performance.

- **Continuous monitoring and anomaly detection:** Evaluate the effectiveness of real-time monitoring systems that track model performance and behaviour. This includes reviewing logs and alerts to ensure abnormal patterns, such as unexpected output shifts or irregular performance, are detected promptly. The system can be assessed according to the ability to flag potential adversarial activities, such as adversarial inputs, data poisoning attempts, or backdoor attacks, in real-time.

- **Environment and deployment assessment:** Assess the isolation and security of the model's deployment environment. This includes ensuring that the model's inference environment is protected from external factors (e.g. physical damage, extreme temperatures, or humidity) and unauthorized access. Regular vulnerability scans and penetration testing can be conducted to identify potential weaknesses in the infrastructure that could be exploited by an attacker.

- **Testing for data sanitization:** Evaluate how effectively the safeguards clean and preprocess input data before it is provided to the ML model. This includes testing for mechanisms that identify and remove noisy, incomplete, or suspicious data that could lead to manipulation of the model. Data sanitization processes should be reviewed and tested regularly to ensure they are effective against threats, such as data poisoning [i.54].

- **Obfuscation of output data:** Assess whether the model's output is appropriately obfuscated to protect sensitive or proprietary information. This could involve testing whether output data reveals any unnecessary details that could be exploited or reverse-engineered. For this, obfuscation techniques like noise addition, rounding, or limiting output precision can be verified for their effectiveness in protecting against information leakage [i.143].

6.8 Security from vulnerabilities

6.8.1 General

Weaknesses of ML models regarding security from vulnerabilities are multifaceted. One the one hand, vulnerabilities can be targeted by information-security related vulnerabilities as well as exploits which compromise an ML model's inference/exploration/exploitation mechanisms. On the other hand, deficiencies in model behaviour-related measures like correctness and robustness as well as bias can induce vulnerabilities, allowing malfunctions [i.36], [i.37], [i.38]. With this background, in the present document, security from vulnerabilities is reflected as the combined fulfilment of requirements regarding model behaviour, information security and safeguards against exploitation. Figure 2 reflects the three core pillars for an ML model's security from vulnerabilities.

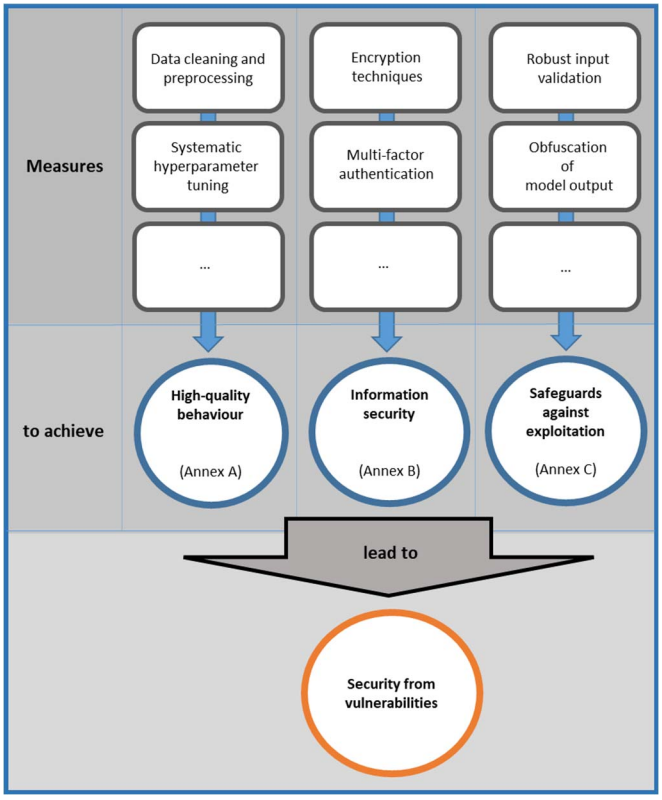

Figure 2: Composing security from vulnerabilities based on three core pillars

6.8.2 Criteria for security from vulnerabilities

The security of machine learning models is a multifaceted challenge that requires the integration of three core pillars: high-quality model behaviour, information security, and safeguards against exploitation. These pillars don't work in isolation but rather complement each other, collectively strengthening the model against vulnerabilities, proceeding from the subclauses of this clause:

- **High-quality model behaviour** lays the foundation for a secure and trustworthy ML-based system. A model's performance is defined not only by its correctness-how accurately it predicts outcomes-but also by its robustness and its ability to handle adverse conditions like noisy data or adversarial attacks. This robustness ensures that the model remains stable even when faced with unexpected inputs or changes in data distribution, making it more resilient to exploitation. For example, a robust model that can handle shifts in input data or adversarial manipulation will be less likely to be tricked by malicious actors trying to compromise its functionality. At the same time, avoiding unwanted bias ensures fairness in decision-making, ensuring that the model does not inadvertently favour one group over another, which might open doors for exploitation in biased predictions (see clauses 6.3 to 6.5).

- With high-quality behaviour, a model is still vulnerable if its underlying data and systems are not properly protected. **Information security** focuses on safeguarding the confidentiality, integrity, and availability of the model, its data, and the systems that support it. For instance, ensuring data privacy prevents sensitive information from being exposed, while model confidentiality protects the proprietary algorithms and parameters from theft. But even if the model's behaviour is top-notch, an adversary could compromise the system by gaining unauthorized access to the model or tampering with its data. This is where authentication and access controls come into play, ensuring only authorized entities can interact with the model. Encryption ensures that data, both at rest and in transit, remains secure from prying eyes, thus protecting the integrity of the model itself from tampering (see clause 6.6).

- Securing the model's behaviour and data makes it the model still be vulnerable to exploits. For protection against exploits, it can be safeguarded through proactive measures. **Safeguards against exploitation** include techniques like input data validation, which ensures only clean, secure data is processed by the model. If malicious data or erroneous inputs were allowed through, even a robust and well-behaved model could be compromised. Similarly, resource management measures like rate limiting and load balancing ensure the model's stability under heavy traffic or attack, protecting against overloads that could lead to system failure. Furthermore, anomaly detection systems monitor the model in real-time to catch potential exploits, such as adversarial inputs or backdoor attacks, before they cause harm. Regarding in- and output data, sanitization ensures that faulty inputs don't undermine the ML model's functionality. Additionally, obfuscation of output data ensures that even if an attacker tries to reverse-engineer the model's decisions, the output remains protected from leakage of sensitive or proprietary information (see clause 6.7).

6.8.3 Assessing security from vulnerabilities

Assessing the security of ML models requires a comprehensive evaluation of each of the three interconnected pillars: High-quality model behaviour, information security, and safeguards against exploitation.

Table 2 summarizes the assessment for all three pillars, according to clauses 6.3 to 6.7.

Table 2: Assessing the three core pillars for security from vulnerabilities

Pillar	Criteria	Description	Assessment
Model behaviour	Correctness	Ensures predictions align with expected outcomes.	Use metrics (e.g. accuracy, precision, MSE) for task-specific evaluation.
	Robustness	Maintains functionality under adverse conditions.	Test with noisy inputs, data shifts, and adversarial examples.
	Avoidance of unwanted bias	Prevents discriminatory or imbalanced outcomes.	Analyse data diversity and fairness in outputs across groups.
Information security	Data privacy	Secures sensitive data during storage and processing.	Assess encryption and anonymization methods.
	Model confidentiality	Protects model architecture and parameters.	Review access restrictions and obfuscation techniques.
	Data/model integrity	Prevents unauthorized modifications.	Test access controls and authentication mechanisms.
	Secure deployment	Protects infrastructure from threats.	Conduct vulnerability scans and penetration tests.
Exploitation safeguards	Input validation	Filters malicious or erroneous data inputs.	Test validation with various input scenarios.
	Resource management	Maintains stability under heavy workloads.	Simulate stress and evaluate rate limiting and load balancing.
	Anomaly detection	Identifies attacks or abnormal patterns.	Monitor real-time performance and detect adversarial inputs.
	Obfuscation	Protects sensitive data in model outputs.	Verify techniques like noise addition or rounding.

6.9 Explainability

6.9.1 Criteria for explainability

6.9.1.1 General

Testing the explainability of ML models requires clarity and understandability regarding ML models' operations. After describing the criteria which are the prerequisites for explainability (consistency of information, human understandability, temporal adaptability of explanations and clarity about mechanisms implemented in an ML model) a questionnaire is presented in Annexes C and D to guide the assessment of an ML model regarding explainability.

6.9.1.2 Consistency of information

The following list outlines key considerations related to the consistency of information as a prerequisite for explaining ML-based systems:

- Rationale of using the ML model (see Annex E);
- Model development procedures (see Annex E);
- Sensitive entities involved into the model development;
- In case of training: statistical distribution of sensitive entities in training data set;
- Model processing procedures (see Annex E);
- Limitations of the ML-based system in terms of the intended use;
- Uncertainty quantification of sensitive entities in inference/exploration/exploitation results;
- Model relevance:
  - Interaction of model with overall system components;
- Overall system around the integrated model;
- Criticality/risk characteristics, e.g. depicting the damage potential on:
  - data;
  - finance;
  - human behaviour (e.g. nudging, sludging, context manipulation);
  - physical and mental well-being.
- Results from conformity assessment procedures.

6.9.1.3 Clarity about ML model methods

Understanding ML model methods implemented into ML-based systems enables to explain:

- How algorithms are used to realize capabilities and execute tasks.
- Which algorithmic characteristics affect which quality criteria (e.g. generalization can affect correctness).
- Constraints in training, inference/exploration/exploitation procedures.

Different Machine Learning (ML) methods have distinct focuses regarding data processing, environmental interactions, and algorithmic characteristics. As a result, customized testing processes for each specific ML method are essential for effectively explaining ML-based systems. In the attempt to achieve explainability as a prerequisite, clarity about the ML methods implemented in ML-based systems makes it possible to design a test setup.

6.9.1.4 Human understandability

The ability to grasp the intricacies, implications, correlations, causes and context surrounding ML-based systems enables stakeholders to explain ML-based systems based on clear information, effectively interpret outcomes, and navigate the complexities associated with ML-based systems. In this context, human understandability depicts the competence of explainers for explaining ML-based systems. Along with clarity about ML methods and consistency of information about ML-based systems, human understanding enables to transform information from a mere foundation into actionable and meaningful knowledge to explain ML-based systems. For the addressed stakeholder, a comprehensible presentation of the explanation is of crucial importance to achieve a clear understanding of the ML-based systems.

6.9.1.5 Temporal continuity of explanations

Continuously explaining ML-based systems across various periods or versions of the implemented ML model is important as those evolve over time, as ML-based systems are subject to updates, retraining, and further modifications. Temporally continuous explainability can be achieved based on the following two aspects:

- Logging: Tracing versions and document changes by maintaining a comprehensive version history enables to trace back and explain the ML-based systems output over time.
- Explanation Consistency: Establishing a homogenous explanation procedure fosters unambiguity for generating explanations.

6.9.2 Assessing explainability

Assessing explainability includes a combination of quantitative and qualitative methods. Feature importance techniques such as SHAP or LIME could be applied to quantify how much each input feature contributes to the model's predictions. Moreover, the use of visual tools, such as heat maps or decision trees, can be evaluated for their effectiveness in making complex modelling decisions accessible and understandable to non-experts. Finally, assessing the model relevance enables to clarify the underlying technical composition of an ML-based systems [i.14].

Annex D provides a questionnaire that has been designed to serve as a guideline for understanding and explaining ML models based on the ML-based systems' impact, risks, conformity with standards, sensitivity considerations, relevance, uncertainty quantification, statistical distribution, system architecture, deployment constraints, and adaptability. This questionnaire can be used to identify the number of questions that can be answered in relation to an ML model in order to understand the extent of explainability.

7 Workflow integration, test methods and definition of test items

7.1 General

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. Akkiraju et al. [i.39] describe a reinterpretation of the Software Capability Maturity Model (CMM) for the machine learning model development process. Amershi et al. [i.40] 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. [i.41] 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:

- In general, in supervised learning, models learn from labelled data to make predictions or decisions. Similar to established practices in software development, supervised learning workflows begin with a comprehensive understanding of business objectives and data requirements. This initial phase, analogous to requirements elicitation and analysis, aims to define the scope of the problem and establish criteria for model success. It involves collaborating closely with domain experts to identify relevant features and labels essential for training predictive algorithms.
- Workflows in unsupervised learning often begin with data exploration and understanding, akin to traditional data preparation in supervised learning contexts. For instance, techniques like clustering and dimensionality reduction are pivotal, aiming to reveal inherent structures within data sets. The workflows in unsupervised learning evolve through stages that merge traditional software engineering practices with the iterative exploration typical of data science. These phases include experimental design, where model architectures and parameters are fine-tuned based on unsupervised metrics rather than direct performance comparisons as in supervised learning loss function analysis.

- Unlike supervised or unsupervised learning, in reinforcement learning agents learn optimal behaviour through interaction with environments. In this context, reinforcement learning workflows emphasize dynamic decision-making and continuous learning in complex, often uncertain settings. Central to reinforcement learning is the training phase, where agents learn through trial and error, optimizing strategies based on feedback mechanisms akin to continuous integration in software development.

Combining workflows and ideas from software engineering and machine learning can provide a solid foundation for developing AI-based applications. In the following, the present document introduces a high-level workflow description, that respects requirements from classical software engineering as well as from ML to support the systematic localization and integration of testing activities. In fact it addresses the main ideas from Machine Learning Operations (MLOps), i.e. taking a holistic approach that combines software engineering, machine learning and operations.

7.2 A workflow perspective for developing and operating ML-based systems

In the context of identifying and locating important quality assurance activities, the present 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 3 shows the complete workflow of a ML model with a supervised learning nature, starting with business understanding, continuing with the ML model development phase and ending up with it's operation and monitoring. The workflow includes classical software development activities, as well as typical data science activities like data preparation, training, and validation. 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.

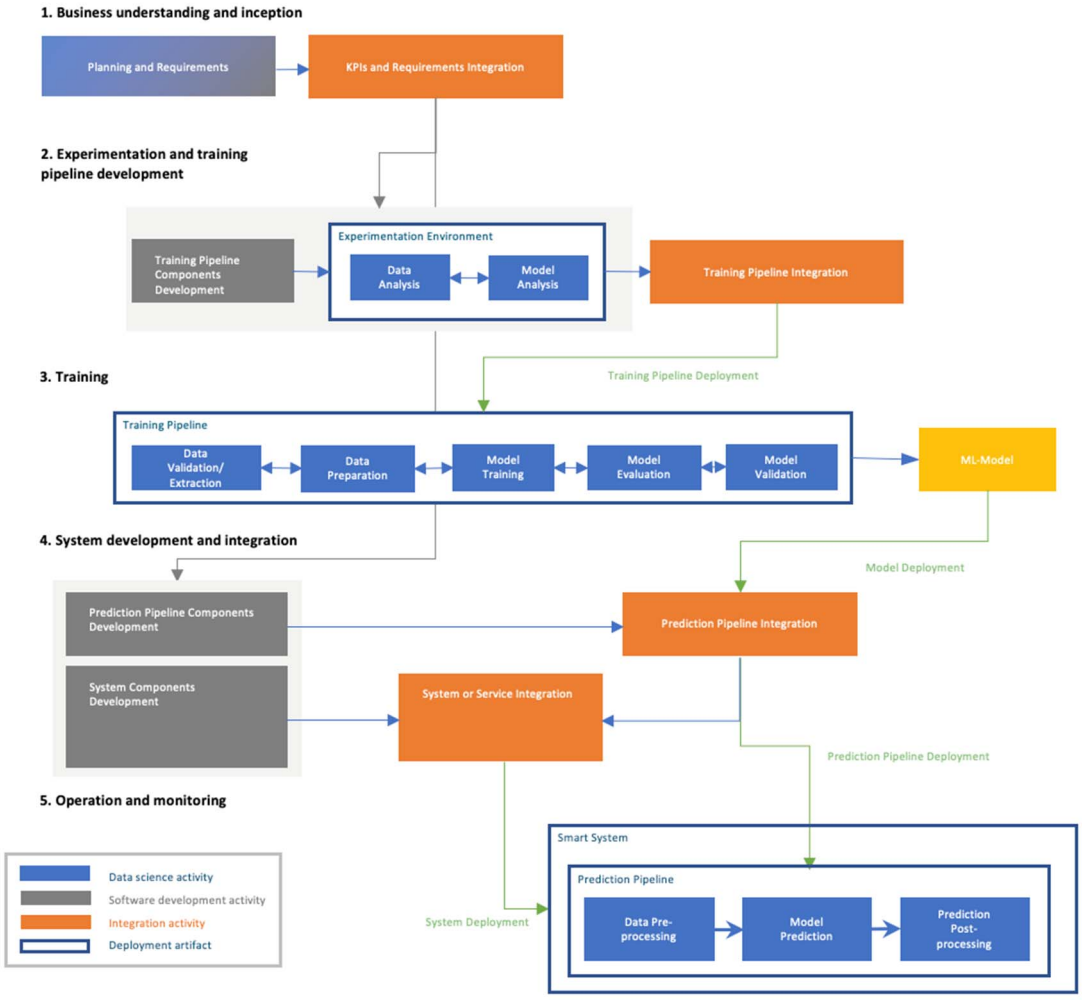

Figure 3: Development and training workflow to develop, train and deploy ML-based systems or applications

On the high-level the workflow distinguishes four different phases that are differentiated and detailed in Figure 3 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).

7.3 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 (Phases 1 - 4 in Figure 3), but can also be performed during operation (Phase 5 in Figure 3). 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, behaviour, 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.

7.4 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, the concept of a test item is used in the following, which includes any work product in the life cycle of an ML-based system, in order to clarify that both static and dynamic test procedures are dealt with. Although the primary test item, as the name of the present document suggests, is the ML-based system, several other test items are obtained 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. [i.3] for example distinguishes on a high-level between testing data, testing the learning program (i.e. the model) and testing the ML-framework (i.e. the libraries and building blocks that are used by a model).

In clause 8, the different test items are systematically derived along the workflow defined in Figure 3. As already mentioned before, test items are normally the work products of a given workflow activity or phase. In this context, a more general overview on important relevant test items is given in the following.

Test items are:

1) **Specifications, requirements and planning documents:** Before an ML-based 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 viewpoints 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, modelling 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, the term pipelines is used 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 should be tested across its integration stages. This includes the individual software components, their partial integration, and the integrated system in the various execution environments. Because 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

8.1 General

Test activities range from testing the individual test items and their integration to larger items in the integration phases. Nearly all phases of the workflow depicted in Figure 3 end with a dedicated integration phase having work products associated that are subject to dedicated testing activities. However, also intermediate work products are of interest for testing. Figure 4 shows the development and training workflow specified in Figure 3 extended by dedicated testing and monitoring activities.

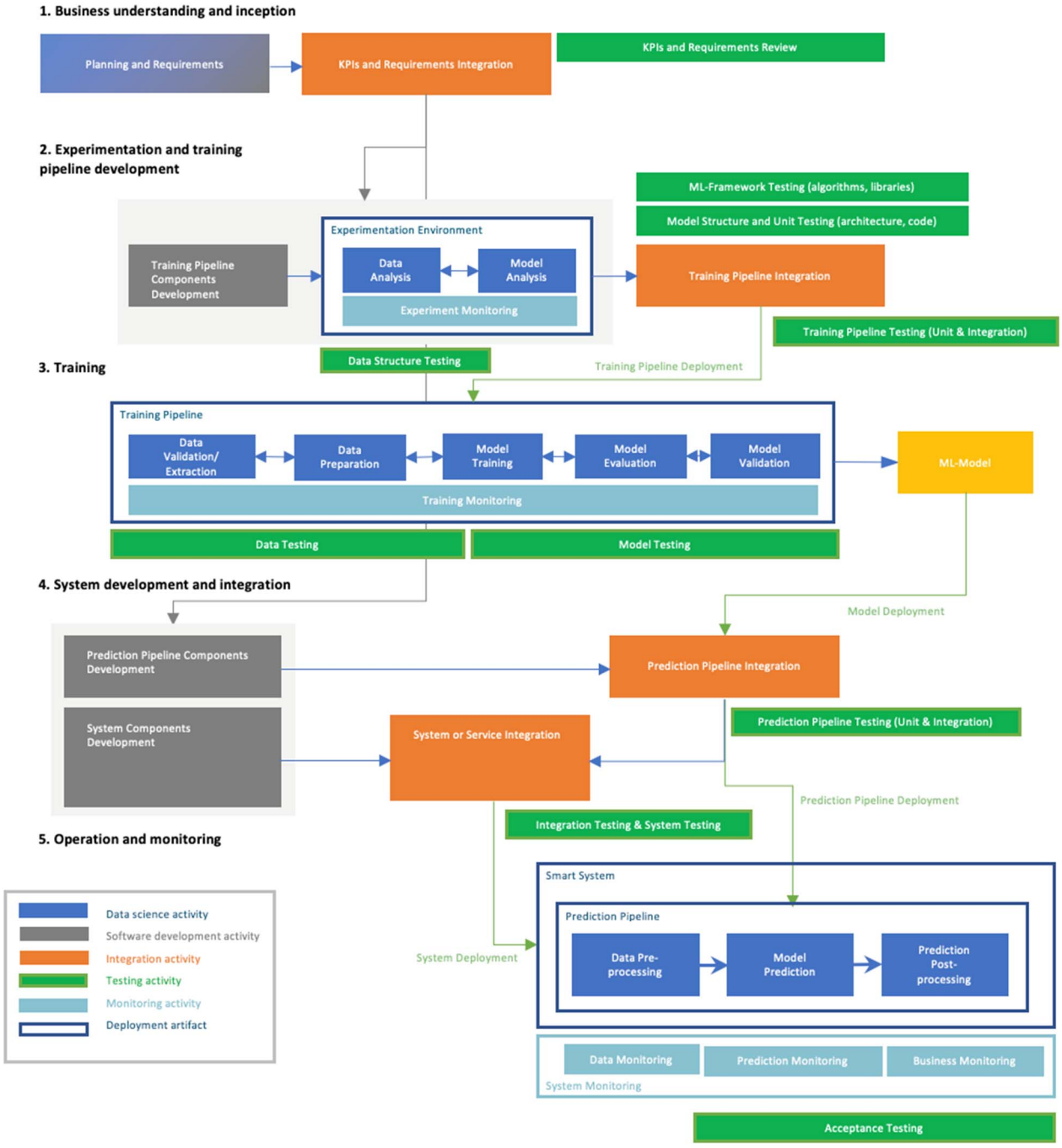

Figure 4: Development and training workflow extended by testing and monitoring activities

Testing activates are denoted in green and monitoring activities are denoted in light blue. 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 they are sometimes the same and show a larger amount of overlap in approaches, methods and results.

The remaining clause 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 activities in Figure 4.

8.2 Test items of the business understanding and inception phase

Figure 5: Testing activities in the business understanding and inception phase

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 harmonisation of KPIs and requirements regarding 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 3 provides an overview on the major work products of the business understanding and inception phase, the related acceptance criteria and items.

Table 3: 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.

8.3 Test items of experimentation and training pipeline development phase

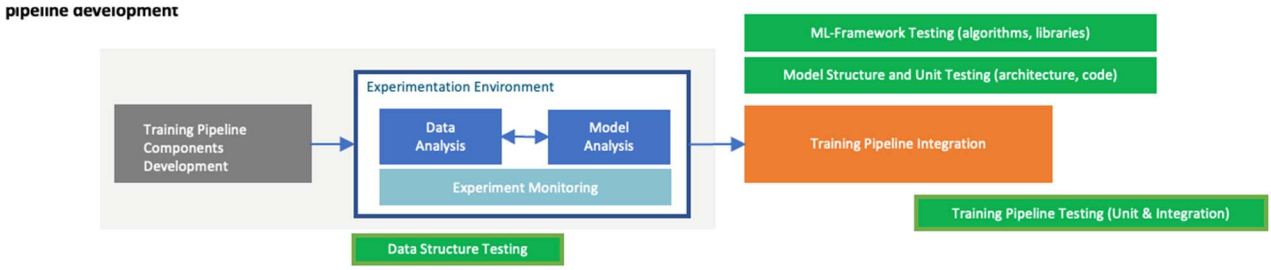

Figure 6: Test activities in the experimentation and training pipeline development phase

The experimentation and training pipeline development phase consists of extensive activities in 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 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 testing 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 4 provides an overview on the major work products of the **experimentation and training pipeline development** phase, the related acceptance criteria and testing types.

Table 4: 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 is available. • Training data is 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 are 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.

Work product/test item	Acceptance criteria	Test type/test objective
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 behaviour during training and inference.
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 unwanted-bias testing. • Testing generalization capabilities (e.g. without distribution data).
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.

8.4 Test items of the training phase

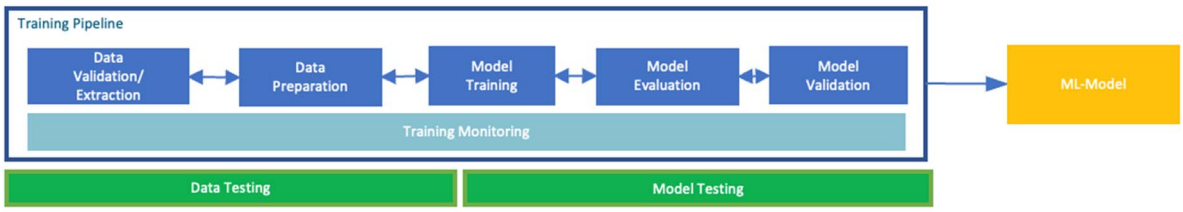

Figure 7: Test activities in the phase of model training

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 final 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 if 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 and deviations, etc.)

Table 5 provides an overview on the major work products of the **training** phase, the related acceptance criteria and testing types and test objectives.

Table 5: 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 is 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. • Testing generalization capabilities (e.g. without distribution data).
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.

8.5 Test items of the system development and integration

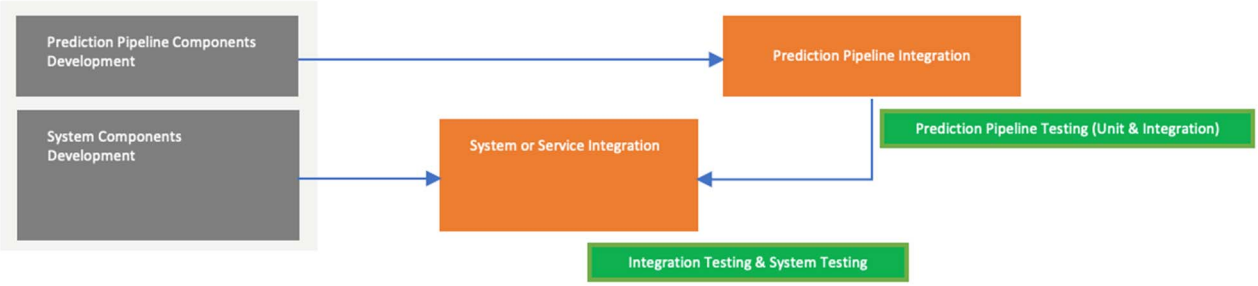

Figure 8: Test activities in the phase of system development and integration

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, the integration of the model with software components that have a direct impact on the quality and performance of the model inference is considered. 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. The result of this integration is called 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 arbitrary 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 6 provides an overview on the major work products of the system development and integration phase, the related acceptance criteria and testing types and test objectives.

Table 6: 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.

Work product/test item	Acceptance criteria	Test type/test objective
Acceptance testing	• The ML-based system complies with stakeholder requirements.	• Performance and stakeholder requirements testing. • Testing the compliance with given rules and regulations.

8.6 The test items of the operation and monitoring phase

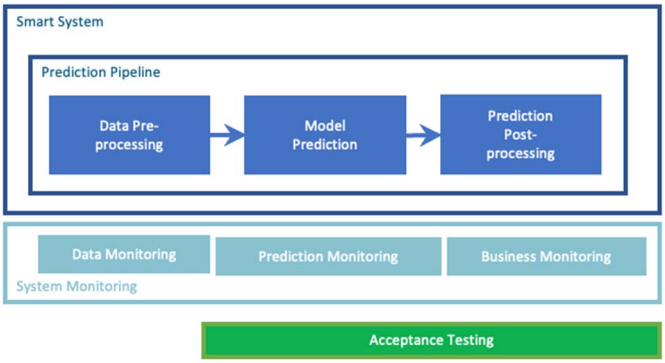

Figure 9: Test activities in the phase of operation and monitoring

For the operation and monitoring phase, the model is executed in its operating environment. Testing and monitoring activities should 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 should 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 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 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 7 provides an overview on the major work products relevant in the operations phase, the related acceptance criteria and testing types and test objectives.

Table 7: 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

9.1 Requirements-based testing

General definition: This most classical test approach relies on available and clearly written requirements, specified for the test item in focus. From those requirements test conditions and test specifications are derived during test analysis and design phases of the conventional software testing process [i.42]. The verification activity aims to proof that a specified requirement is fulfilled by a successful execution of the corresponding implemented test cases and meeting the defined expected behaviour and results.

The quality of the requirements is essential to be able to derive useful and effective test cases. In particular each requirement should meet quality criteria, as e.g. defined in [i.45]. Each requirement should be e.g.: *unique*, *necessary*, *consistent*, *complete*, *testable*, and *realizable*. Additional quality criteria may be expected like *understandable* or *confirmed*. Requirements may cover functional as well as non-functional quality aspects [i.43], [i.44].

For ML-based systems explicit and sufficient functional requirements do usually not exist (see clause 5.1), as typically the systems behaviour is described by data examples instead of the underlying patterns. Nevertheless, it is still possible to identify functional and non-functional requirements. Those may cover quality characteristics [i.43] like functional performance, computational, energy or memory efficiency, flexibility or adaptability, just to name a few.

How it works: Prerequisite of this analytical approach is a set of specified and agreed requirements which should be applicable to exactly the test item in focus. The analysis of these requirements includes a review of the requirements with respect to the expected quality criteria (e.g. [i.45]). The reviewed requirements are potentially broken down into a set of distinct test conditions, from which then high-level test cases are derived. By identifying specific input and output values for the required test data low-level test cases are further defined.

During the process of analysis, break down and derivation of test cases all artifacts (test items, requirements, test conditions, and high-level test cases) are linked to each other (e.g. by referring to individual IDs) such that all items become traceable in any direction.

Due to the probabilistic nature of ML-based systems the definition of low-level test cases is however not straightforward. A single unexpected result or behaviour of an ML-based system does not necessarily indicate a failure of the system. It is important to assess the behaviour as a statistical process and measure the actual probabilities and compare them e.g. against the functional performance metrics like *accuracy* or *precision*. The challenge in the assessment of the ML-based system is to understand the completeness and variations of the important parameters in the system context (e.g. for an image recognition system using an outdoor camera: the illumination, weather conditions, viewing angle, camera focus, a.s.o.).

Types of issues addressed: The approach of requirements-based testing allows to discuss, review and improve requirements for ML-based systems, even if most often there are hardly any strong requirements in ML projects in the beginning. Moreover, it combines well with the risk-based testing approach, which is discussed in clause 9.2.

The decision to use requirements-based testing also fosters the discussion of the acceptance criteria for an ML-based systems functional performance or non-functional quality criteria in an early stage of a project.

9.2 Risk-based testing

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 should 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 [i.33], [i.34] and [i.35].

How it works: Risk-based testing is a pragmatic and often intuitively used approach [i.47] to focus test activities on those scenarios that trigger the most critical situations of a software system. See Erdogan et al. [i.48] for a comprehensive survey of risk-based testing approaches and [i.49] for a systematic compilation of different approaches to risk-based testing in the context of IT security [i.52]. 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 should 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 behaviour 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 consider functional deficiencies and their relation to mission and business criticality.

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 [i.50]. However, even though ISO 21448 [i.11] recommends the combination of risk assessment and testing no systematic approach is yet described. In [i.51] 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. [i.53] 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.

9.3 Search-based testing

General definition: Search-Based Testing (SBT) is the application of optimizing search techniques to solve software testing problems. 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 can 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 used in various ways for testing and validating ML models or other artefacts in the ML lifecycle. For example, search-based algorithms are suitable for generating diverse and comprehensive input data sets for testing ML models. This applies to the generation of synthetic test data, data augmentation for testing and exploration of the test data space to ensure better coverage of input variations. The goal can be to uncover potential decision boundaries or identify parts of the ML-based system, such as certain features or dataset characteristics, that are most responsible for poor performance [i.8]. This is crucial to assess how well the model performs in different scenarios, including selected borderline cases. In addition, SBT can be used to efficiently search for such negative examples [i.55] to identify weaknesses and improve the robustness of the model. In load and performance testing, searching for test cases that push the ML model to its limits, such as processing very large inputs or inputs with extreme values, can be used to evaluate its performance under stress or to determine how a model performs under different amounts of data and speeds. Finally, SBT techniques can also be applied to reinforcement learning settings e.g. to optimize the agent's behaviour or policy.

9.4 Combinatorial testing

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 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 order of combinations (3-wise, 4-wise etc.) 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 [i.56] 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. However, the relative frequency of failed vs. passed runs is not an appropriate quality measure. It should be weighted by the (Radon-Nikodym) derivative relating the uniform distribution behind the combinatorics and the empirical one.

9.5 Probabilistic testing

General definition: The term probabilistic testing describes test methods and approaches used to evaluate the performance and reliability of systems under variable and uncertain conditions. These types of tests use theories of probability and randomness to understand how systems will behave in a variety of possible usage scenarios that they may encounter in the real world [i.57]. It has principal relations to the idea of risk-based testing described in clause 9.2.

How it works: Probabilistic testing uses stochastic modelling methods, stochastically generated inputs and statistical analysis techniques to define and evaluate tests. Systems and/or their environment are modelled using stochastic processes that help to represent complex behaviours and interactions under uncertainty. These models simulate the various probabilities of different system or environment modes and transitions between these modes based on defined probabilistic rules. Pietrantuono and Russel [i.58], for example, use probability sampling that considers additional information about the software under test to assess reliability in an unbiased and efficient way.

Probabilistic tests are typically designed to include random inputs or simulate random events to assess how the system handles and responds to unforeseen or varying conditions. In addition, statistical methods are used to analyse the test results. These include calculating probabilities for specific outcomes, using statistical significance tests to determine confidence in test results, and applying models such as Bayesian inference to make informed predictions about system performance under uncertainty. Key performance metrics include reliability, availability, and fault tolerance. These metrics are evaluated under the assumption that system inputs and environmental conditions can vary according to known probability distributions.

Types of issues addressed: In the context of ML, probabilistic testing offers a systematic way of dealing with complex probability distributions in the state space of the application or the application environment. In order to understand the complex dependencies between the environmental conditions of ML-based object recognition, Wiesbrock and Grossmann [i.59], for example, introduce the concept of a probabilistically extended ontology, which combines traditional ontologies with probabilistic models. These ontologies are then used as a baseline for deriving tests and evaluating test results so that stochastic hypotheses about the application environment can be considered and a more realistic test statement can be made compared to an assumed uniform distribution of events.

9.6 Metamorphic testing

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 [i.61]. 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 (MRs) 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, the output of the original input data and the transformed versions are compared with each other. 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 [i.62].

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 mislabelled image will pass a Metamorphic Test without exposing the mislabelling.

9.7 Differential testing

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 [i.63].

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 [i.64]. 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 [i.65].

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.

9.8 Testing by Adversarial Attacks

General definition: Adversarial Attacks refer to the subtle modification of original inputs to a trained ML 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 [i.66]. 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) $x' = x + \eta$ 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 [i.67]. To mitigate this risk, Adversarial testing 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 [i.68]. A more comprehensive overview on adversarial attacks including different tools and techniques is given in [i.1].

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 ML models, as well as to identify potential areas for improvement in their design and implementation.

9.9 Reviews

General definition: 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: In the context of ML, reviews can be performed on various components such as the code (code review), data (data review) or the model (model review/validation). During Code review, expert reviewers analyse the source code of an ML-based system to identify any programming errors, inefficiencies or security vulnerabilities. This step is essential for ensuring the robustness and maintainability of the codebase. In Data review, reviewers scrutinize the data to detect any biases or anomalies that may affect the model's performance. Ensuring a diverse and representative dataset is crucial for preventing biased outcomes and promoting fairness in ML-based systems. Model review/validation involves assessing the model's performance against predefined criteria. This includes validating the model on different datasets, evaluating its accuracy, precision, recall, and other relevant metrics to ensure it meets the desired standards. For instance regarding the European single market, ML-based systems should adhere to legislative documents as well as regulation-related technical standards. Reviews are applied to assess compliance with relevant laws, industry, regulations, and data protection requirements. Ensuring that ML-based systems operate within the ethical and legal frameworks governing their use.

Types of issues addressed: In traditional software engineering requirements, design, code, user interface are subject to reviews. 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 training process. Moreover reviews can target different quality attributes like performance, bias and fairness, Explainability and interpretability, and 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.

9.10 Static analysis

Static analysis refers to the examination of code, models and data without actually running the system. This technique can be particularly useful for testing in the context of ML-based systems to ensure the reliability, safety robustness and efficiency of ML-based systems. There are several ways in which static analysis can be used to test ML-based systems.

How it works: In contrast to reviews, static analysis is an automated process in which the code, model and data of the ML-based system are analysed with the help of tools without being executed. The basis for the analysis is formed by predefined test rules and algorithms that can be executed as often as required to analyse the test object. Static analysis is therefore well suited for integration into automated pipelines and can be conducted with less specialized knowledge, as it relies on automated tools and predefined rules. The results are easily comparable but may show a larger number of false positive issues. Incorporating static analysis into the development and maintenance process of ML-based systems can significantly improve their quality, security, and reliability. However, it's important to note that static analysis is just one part of a comprehensive verification strategy and should be complemented with dynamic analysis, testing, and other quality assurance practices.

Types of issues addressed: Static analysis tools can be used to examine the source code of ML-based systems and the model code for common programming errors, adherence to coding standards, and potential security vulnerabilities. This includes checking for buffer overflows, memory leaks, and other issues that are common in software development. It can help identify outdated or vulnerable libraries and other dependencies that may pose security risks or compatibility issues. It can also be applied to the datasets used for training and testing ML models. This may involve checking for imbalanced data, missing values, outliers, or other issues that could affect the performance or fairness of the model.

For ML-based systems used in regulated industries, static analysis can help ensure that the system complies with relevant standards and regulations. This includes checks for data privacy, security, and other regulatory requirements. Finally, by examining the data and model structure, static analysis can help in identifying potential ethical issues and biases in ML models, ensuring that the systems are fair and do not discriminate against certain groups.

9.11 A/B Testing

A/B testing, also known as random controlled experiment, is statistical method used to compare two variants (A and B) of a specific element or feature in a controlled environment with the purpose to determine the most effective variant among the options being tested.

How it works: Define the Hypothesis: before starting the test, a hypothesis should be formulated. This hypothesis often takes the form of predicting the expected impact of a particular change. For example, in ML-based systems, the hypothesis may be that a change in the underlying learning algorithm will increase the overall system performance and user satisfaction.

Create variations: Two or more variants (A, B, C, etc.) are created, each representing a different version of the element being tested. In machine learning, this could mean different learning algorithms.

Conduct the test: First of all, participants, users, or data points are randomly assigned to each variant. The randomization helps ensure that the samples are statistically representative and reduces biases. The test is then run for a specific period, during which the system collects data on performance metrics, or any other relevant measurements.

Analyse Results and Draw Conclusions: After the test period, the collected data (ML performance metric) is analysed using statistical methods to determine the significance of any observed differences between the variants. Based on the analysis, conclusions are drawn regarding which variants performed better or worse in achieving the desired outcome. The results may support or reject the initial hypothesis.

Types of issues addressed: A/B testing may be used to address both functional and non-functional issues. For example, in machine learning, functional issues may involve determining which underlying learning algorithm improves the system's predictive accuracy. While non-functional issues may include evaluating the computational efficiency of different learning algorithms or comparing their inference time.

10 Challenges in testing ML-based systems from the perspective of the test process

10.1 General

In the following clauses, challenges related to the different phases of a test phases are outlined. This includes defining quality objectives, determining test items, and creating a tailored test strategy to address unique ML challenges like data errors and bias during the test planning phase, identifying appropriate data testing procedures and coverage criteria during the test design phase, automation challenges during test execution as well as ensuring comprehensible communication of results and addressing ML-specific issues like non-determinism and robustness during test evaluation and reporting.

10.2 Test Management for tests of ML-based systems

Test management comprises the planning, scheduling, estimating, monitoring, reporting, control, and completion of test activities within the testing process [i.42].

Test management for ML applications requires a shift in focus from traditional software testing methods. It demands a deeper understanding of data science principles, a flexible and adaptive testing approach, continuous monitoring, and a strong emphasis on ethical and fairness considerations.

In this clause, challenges related to overall test management aspects are outlined. In particular, test selection and prioritization to maximize the effectiveness of testing under resource constraints is a universal challenge in test management, amplified by the specifics of the development of ML-based systems. Challenges specific to the individual phases of the testing process for ML-based systems are outlined in the subsequent clauses.

Challenge 1: Labelling efforts for test data

ML-based systems have a huge input space, that requires broad spectrum of test data to cover adequately. Preparing test data for ML-based systems requires extensive annotation and labelling.

Test selection and test prioritization aims to optimize the testing efforts for maximum effectiveness with the available resources. Due to the non-deterministic nature of ML-based systems and the impact of training data and parameter selection, as well as the substantial resources needed for training and operating ML-based systems, test selection and prioritization needs to be considered very thoroughly throughout the testing process. Changes to the training data or parameters may invalidate existing tests, thus the test selection and prioritization needs to be re-evaluated and adapted continuously.

Traditional methods for the test selection and prioritization are often risk-based, where risks are identified, assessed, and mitigated, as part of the risk management strategy. Such a risk-based approach is outlined in clause 9.2 and implies that the testing efforts and resource use are proportional to the identified risks. Test prioritization and selection serves to allocate the available resources efficiently in order to mitigate the most substantial risks. Depending on the context, different levels of risk assessment may be employed, ranging from light weight to more formalized approaches.

Early assessment of risks is essential as it may influence the overall development and workflow as well, for example the data sourcing and preparation. Considering the substantial resources needed for the training of ML models as well as potential cascading effects along the development process, early testing and risk mitigation activities are essential in optimizing resource use. Thus, this should be taken into account during the test planning, e.g. to also establish adequate entry and exit conditions for the different test items outlined in clause 8 as well as the evaluation and interpretation of test results and their implications for the overall workflow. Corresponding traceability links and documentation need to be provided to support the different activities.

Test selection and prioritization assumptions need to be updated as needed, or even periodically to ensure that they are aligned with the test goals and strategies, as well as upstream changes in training data and parameters. The inherent non-determinism also necessitates frequent validation of the test selection and prioritization. The different kinds of biases also need to be considered during the test selection and prioritization, in conjunction with associated risks.

In the research literature, attempts have been made to identify further test selection and prioritization criteria, specific to different kinds of ML-based systems. Wang et al. [i.69] conducted a systematic review of research on test input selection and prioritization for DNNs. Test prioritization techniques can be categorized as coverage-based, confidence-based, and mutation-based (Dang et al. [i.70]). Mosin et al. [i.71] compared different test input prioritization techniques regarding their effectiveness and efficiency. Dang et al. [i.70] proposed a test prioritization approach for classical ML models, leveraging model interpretability and engineered features. Demir et al. [i.72] proposed a distribution aware DNN testing framework that prioritizes test data based on potential to cause incorrect predictions.

10.3 Test process for ML-based systems

10.3.1 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 outlines which parts of the system will be tested, the intensity of each test, the methods and techniques to be applied, the test infrastructure to be used, as well as the sequence in which the testing will occur.

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

Challenge 1: Selection of appropriate quality criteria and test items

Since ML-based system slightly differ in terms of engineering as well as operation, the test process should address additional test items, 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 e.g. 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 should be considered that are given little to no attention in classic software. These test items are:

- data and labels;
- hyperparameters;
- loss function;
- optimizer;
- 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 ensure 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 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. [i.80], which considers the activation of individual neurons in a network as a variant of statement coverage. Ma et al. [i.81] 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. [i.82] 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. [i.83] and Cheng et al. [i.84] 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. [i.85] 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 [i.86] and [i.1]. In addition, the work of Dong et al. [i.87] 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.

Moreover, clause 8 provides guidance on how test items, test approaches and appropriate acceptance criteria can be systematically mapped with each other.

Challenge 4: 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. Taking 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, including ML-Model, data pre-processing, decision making;
- integration testing, including "model-in-data" pre-processing chain, "model-in-data" pre-processing chain + decision making, model subsystem with safe-guarding;
- system testing, including the entire system in test environment;
- acceptance testing, including the entire system in operational environment;
- runtime testing during operation.

10.3.2 Test Design and 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 as wee as 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 should be systematically tested and reviewed.

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 should 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 [i.73], [i.74]).

According to L.P. English [i.75] 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.

The identification of relevant quality attributes forms the basis for the assessment and subsequent improvement of data quality. The attributes 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 [i.76].

- 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 pre-processing 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.

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

Testing ML suffers from a particularly difficult form of the oracle problem. While classical systems are usually fully specified, ML-based systems are designed to provide meaningful answers to questions for which there is no known answer yet [i.3] (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 should 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 should 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 [i.46], 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.

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 is expected or desired in the data when manually selecting and labelling data.

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.

Lack of 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 acceptable 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. Such distributional differences can be considered as gaps in the representation of reality and are a general problem in designing ML models 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: Complex ML models 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 ML is generally referred to as explainability, which includes both interpretability and confidence in the system and its genesis [i.77], [i.78]. While interpretability is the degree to which a human can understand the cause of a decision [i.79], 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 quality properties of neural networks, it is essential to enable at least a certain degree of human interpretability and understandability.

10.3.3 Test Implementation and 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-based 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 missing interpretability 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 trustworthy and reliable manner.

However, generating test cases automatically is still a challenge. According to Zhang et al. [i.3] studies claimed that the test cases generated by an automated testing tool may not cover all real-world cases.

10.3.4 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

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 may worsen the KPIs of another property.

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.

To support the definition of appropriate metrics and acceptance criteria, clause 6 introduces the notion of quality attributes and relates this attribute with appropriate quality metric. In addition, clause 8 introduces test methods and related acceptance criteria to ease the definition of exit criteria for testing.

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 should 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.

Annex A:
Assessing **correctness, robustness, avoidance of unwanted bias** of ML models (potential risk sources for the criterion "security from vulnerabilities")

A.1 Causes related to usual model behaviour

A.1.1 Cause: Noise and outliers in inferred/explored data

Noise refers to random errors or fluctuations in data that obscure patterns, often due to measurement errors, environmental variations, or sensor inaccuracies. Outliers, on the other hand, are data points that deviate significantly from the modus/norm, which can result from rare events, data entry errors, or variability in the data population itself. Both noise and outliers can distort model training and predictions, able to reduce generalizability and correctness.

Measures to detect	ML method
• Cluster Stability: The stability of clusters can be assessed using metrics like the Jaccard index, Adjusted Rand Index (ARI) or Adjusted Mutual Information (AMI): These metrics compare the clustering result against some ground truth, helping to identify if outliers are affecting the overall agreement with the expected clustering. • DBSCAN Density-based clustering can identify noise as points that do not belong to any cluster [i.105]. • Silhouette Score: Measures how well-separated clusters are. A lower score may indicate the presence of noise or outliers affecting cluster formation [i.134].	SL/UL
• Examining the sensitivity of the policy to small perturbations or noise in the environment, [i.95], [i.96].	RL

Countermeasures	ML method
Data cleaning and preprocessing [i.103], [i.104], [i.105], [i.106]: • Identify and removing outliers as well as noise-specific data points before applying the clustering algorithm can improve the silhouette score [i.134]. • Try algorithms less sensitive to noise and outliers, e.g. like DBSCAN in UL, which explicitly mitigates noise points, [i.90]. • Apply noise reduction techniques, such as filtering or smoothing, to enhance the robustness of the model against noisy data during inference. • Use of regularization techniques, such as L1/L2 regularization, or ensemble methods to increase the model's resilience against unexpected or outlier data [i.135].	SL/UL
• Use of RL methods that are less sensitive to environmental noise or perturbations and handle noise more effectively, e.g. Dueling DQN or Noisy Nets [i.107]. • Noise can be filtered out from state inputs, using smoothing as well as filtering mechanisms for state inputs to reduce the impact of noisy observations, e.g. via Kalman filters or moving averages, [i.108]. • Incorporation of multiple policies to aggregate decisions from multiple policies can mitigate the relative impact of noise.	RL

A.1.2 Cause: Curse of dimensionality

The curse of dimensionality occurs when an increase in the number of features reflect dimensions and lead to a high (e.g. exponential) growth in data volume, making it sparse and challenging to analyse. This sparsity can weaken model performance if patterns become harder to detect, whereafter models could require significantly more data to achieve reliable generalization. Additionally, high-dimensional data can cause overfitting, as models may capture noise rather than meaningful patterns.

Measures to detect	ML method
• Variance Ratio: Metrics like the explained variance ratio in Principal Component Analysis (PCA) can help understand how much information is retainable by a model after changing the data space dimension. Changing dimensions without significantly impacting the correctness of a model can indicate an increase in robustness. • Manifold Learning Techniques: Metrics such as t-distributed Stochastic Neighbour Embedding (t-SNE) or Uniform Manifold Approximation and Projection (UMAP) may reveal by visualizations if high dimensionality is affecting the separability of data [i.91], [i.92], [i.93].	SL/UL/RL

Countermeasures	ML method
• Choice of resilient clustering algorithms designed to be less sensitive to variations in sensitive entities, such as DBSCAN [i.105] or OPTICS that can handle varying densities and shapes of clusters.	UL
• Sensitive entity-specific feature engineering to reduce the impact of variations in sensitive entities on the clustering process. This can be achieved via removing noisy or irrelevant features or including synthetic data for underscoring specific patterns. • General data pre-processing techniques as normalization and standardization to reduce the impact of variations in the data. For instance, standardizing features and data transformation to lower-dimensional spaces can help in reducing the influence of entities with high variances.	SL/UL/RL

A.1.3 Cause: Poor configuration of hyperparameters

Poor configuration of hyperparameters, such as learning rate, batch size, or regularization strength, can lead to a low model performance, as improper settings may cause underfitting or overfitting. Misconfigured hyperparameters, can also slow down training, cause convergence issues, or lead to unstable predictions. In case of no hyperparameter tuning, the model's flexibility is lowered, making it harder to balance complexity and generalization for accurate and efficient learning.

Measures to detect	ML method
• Analyse how sensitive the model is to changes in hyperparameter values, identifying critical parameters that significantly affect performance and result in vulnerabilities. • Assess the model's response to intentionally manipulated inputs that target potential bias for different hyperparameter choices.	SL/UL/RL

Countermeasures	ML method
• Use systematic hyperparameter tuning procedures such as grid search, random search or Bayesian optimization to identify optimal hyperparameter configurations [i.108], [i.109]. • Implement automated hyperparameter tuning tools or frameworks to systematically explore the hyperparameter space and optimize model performance. • Conduct sensitivity analysis to understand how changes in hyperparameter values impact model predictions and unwanted bias metrics. This information can guide the selection of hyperparameters that minimize bias. • For each hyperparameter choice, introduce adversarial examples specifically designed to exploit or reveal bias in the model. These examples may involve perturbations in the input data that highlight vulnerabilities related to sensitive entities, [i.110].	SL/UL/RL

A.1.4 Cause: Over-reliance on local patterns

Over-reliance on local patterns occurs when a model focuses too much on small, specific details of the training data, rather than capturing broader, more generalizable trends. This can lead to overfitting, where the model performs well on the training data but fails to generalize to new, unseen data. It reduces the model's robustness, making it sensitive to noise or small fluctuations that do not reflect the decisive underlying relationships in the data.

Measures to detect	ML method
• Assess the generalization gap to identify instances where the model relies too heavily on local patterns without effectively generalizing, what can be exploited by adversarial inputs [i.94]. • Identify instances where the model overfits to specific local patterns in the data, compromising its ability to make accurate predictions on unseen data.	SL/UL/RL

Countermeasures	ML method
• Use cross-validation to assess model performance across different subsets of data, preventing overfitting to local patterns [i.112], [i.136]. • Periodically assess model performance and behaviour on the basis of adversarial assessments to identify instances of overfitting or over-reliance on specific patterns. Adjust the model or training process accordingly. • Flagging, analysis and extraction suspicious data points/portions during data pre-processing enables to remove anomalies in the dataset to reduce the impact on sensitive entities.	SL/UL/RL

A.1.5 Cause: Inadequate sampling of diverse data points

Inadequate sampling of diverse data points occurs when the training data lacks sufficient variety or representation of different scenarios, leading to inaccurate and/or biased models. This can result in poor generalization, as the model may perform well on the overrepresented data but struggle with new, unseen variations. Without adequate diversity in the training set, the model may fail to capture essential patterns and relationships, limiting its ability to make accurate predictions across a broad range of inputs.

Measures to detect	ML method
• Measure the ratio of favourable outcomes for different entities and evaluate the difference in true positive rates across different entities.	SL/UL/RL

Countermeasures	ML method
• Balance classes of sensitive entities by stratified sampling/adding synthetic data points to ensure that each subgroup in the dataset is represented proportionally. • Use oversampling for underrepresented groups and undersampling for overrepresented groups to balance the dataset [i.111].	SL/UL/RL

A.1.6 Cause: Inadequate data pre-processing stages

Inadequate data pre-processing can lead to poor model performance, as raw or improperly processed data may introduce inconsistencies, errors, or irrelevant features. This includes issues such as missing values, incorrect data types, or unscaled features, which can confuse machine learning algorithms and hinder their ability to learn effectively.

Measures to detect	ML method
• Examining the sensitivity of the policy to small perturbations or noise in the environment. • Kolmogorov-Smirnov statistic: examine the cumulative distribution functions of predictions for sensitive entities to identify differences [i.95]. • Correlation analyses regarding sensitive entities in the data set [i.94].	SL

Countermeasures	ML method
• Data Exploration and Profiling: Conduct a detailed analysis of the dataset to identify issues like missing values, outliers, and inconsistent scales before starting preprocessing. This ensures a clear understanding of what preprocessing steps are necessary.	SL

A.1.7 Cause: The developer's preferences/incomplete understanding of task/domain

When a developer's preferences or incomplete understanding of the task or domain influence model design, it can lead to biased assumptions, inadequate feature selection, or suboptimal model choice. This misalignment with the real-world problem may cause the model to miss critical patterns or fail to address the true objectives of the task. Without comprehensive domain knowledge, the developer may overlook essential data or nuances, reducing the model's relevance, accuracy, and effectiveness in practical applications.

Measures to detect	ML method
• Calculations of disparate impact, statistical parity, and equalized odds help measure biases in model predictions across different sensitive entities to highlight disparities, [i.96]. • Confusion/matching matrix analysis: Examining the confusion matrix can reveal biases in the model's predictions across different sensitive entities. In this context, disproportionate misclassifications may indicate bias in predictions.	SL/UL/RL

Countermeasures	ML method
• Involve stakeholders, including end-users and domain experts, in the development process. Regular feedback from diverse perspectives helps refine the model and ensures a more accurate representation of the task or domain.	SL/UL/RL
• Use synthetic data points to augment the data set to increase the task/domain-dependent representativeness and consistency of the sensitive entities.	RL

A.1.8 Cause: Reward deviation

Reward deviation occurs when the actual rewards received by the model diverge from the intended reward structure, often due to poorly defined reward functions or external inconsistencies. This misalignment can cause the model to learn unintended behaviours, optimizing for objectives that do not accurately reflect the desired outcomes.

Measures to detect	ML method
• Measurements of deviations from expected reward pattern. For instance, using the Bayesian control rule, deviations can be quantified and assessed in relevance according to the model's beliefs [i.97].	RL

Countermeasures	ML method
• Reward distributions can be smoothened and constrained via regularization, while mitigating extreme rewards that deviate from expected patterns [i.113]. • Normalization of rewards or customization of reward functions can prevent extreme deviations or inconsistencies in reward patterns. Techniques like min-max scaling can ensure rewards are within specific bounds [i.114].	RL

A.1.9 Cause: Inadequate exploration-exploitation trade-off

An inadequate exploration-exploitation trade-off arises when the model either explores too much (testing new actions without leveraging learned information) or exploits too much (relying on known actions without seeking potentially better options). Poorly balancing this trade-off can lead to suboptimal solutions: excessive exploration can waste resources and slow convergence, while excessive exploitation may cause the model to miss better outcomes.

Measures to detect	ML method
• Measuring the entropy can be depicted the information gain or uncertainty. Higher entropy can indicate that the model is exploring multiple possibilities, but it also indicates randomness, impacting model correctness [i.98]. • Monitor the diversity of the system's reward over time. Low diversity indicates excessive exploitation, while erratic or unstructured outputs suggest excessive exploration.	RL

Countermeasures	ML method
• Balancing of exploration and exploitation [i.115]. • Adaptation of context-dependent statistic criteria, while the model learns. • Implement adaptive exploration techniques like epsilon decay (gradually reducing exploration over time) or Upper Confidence Bound (UCB) to allow dynamic adjustment based on the environment. • Combine multiple exploration techniques, such as random exploration and informed exploration, where the model intelligently explores based on domain knowledge or uncertainty. • Continuously monitor how feedback from exploitation or exploration affects system behaviour and adjust accordingly to ensure the model isn't stuck in an exploration-only or exploitation-only cycle.	RL

A.1.10 Cause: Context-dependency of characteristics

Context-dependency of model characteristics refers to designing a model to perform optimally only within a specific context or set of conditions, which can limit its ability to generalize to other scenarios. This context-dependence can lead the model to underperform or make incorrect predictions when faced with data outside its intended use case.

Measures to detect	ML method
• Assess the correlation of sensitive entities with a ground truth dataset in different contexts. • Test the model on data that lies outside its training distribution to observe how its performance degrades. This highlights the boundaries of the intended context and uncovers reliance on specific data patterns. • Introduce controlled variations to input data, such as shifts in distribution, noise, or adversarial perturbations, to determine whether the model maintains performance under altered conditions.	SL

Countermeasures	ML method
• Introduce constraints for biased outcomes that are sensitive to different contexts. This ensures that model decisions are consistent and unwanted bias is mitigated across various situations [i.116]. • Enrich the training dataset with diverse examples from related but distinct contexts to encourage the model to learn generalized features.	SL

A.1.11 Cause: Erroneous feedback loop

An erroneous feedback loop occurs when incorrect outputs from a model are fed back into the learning process as if they were correct, reinforcing mistakes and leading to compounding errors over time. This issue can degrade model performance, as the model increasingly optimizes around flawed information rather than accurate patterns. Left unchecked, erroneous feedback loops can cause the model to drift far from intended outcomes, undermining its reliability and usefulness in practical applications.

Measures to detect	ML method
• Deliberately alter model outputs and observe the consequences on the feedback loop [i.117]. • Continuously track key performance metrics across different data segments and time periods. Sudden or steady performance degradation may indicate errors.	SL

Countermeasures	ML method
• Incorporate bias constraints directly into the learning as well as inference/exploration/exploitation process to guide the model's output towards more equitable outcomes. • Incorporate mechanisms to validate and filter feedback before incorporating it into the system. • Separate the training process from real-time feedback loops by maintaining a static model during live operations and updating it periodically with vetted data.	SL/RL

A.1.12 Cause: Faults training data

Faults in training data, such as noise, bias, imbalance, or insufficient quantity, can hinder a model's ability to learn accurately and generalize effectively. Poor-quality data may cause the model to pick up incorrect patterns or develop biases, leading to inaccurate predictions and poor real-world performance.

Measures to detect	ML method
• Missing Values Rate: Quantifies the percentage of missing values in the dataset, affecting the model's ability to learn from incomplete data. • Outliers Detection: Metrics like Z-Score, Interquartile Range (IQR), or Mahalanobis distance help identify anomalies or outliers. Outliers can skew the learning process and impact the model's generalization [i.137]. • Class discrepancy: Statistical measures such as Kolmogorov-Smirnov test or Kullback-Leibler divergence can be used to assess differences in data distributions. Existing discrepancies could affect the model's assumptions and lead to incorrect predictions. For instance, high imbalance in data can lead to bias or poor generalization towards the minority class. Moreover, measuring the amount of information or uncertainty in the dataset, e.g. high entropy, can indicate a lack of clear patterns which may affect the ML model's prediction performance [i.99], [i.100]. • Involve human annotators to identify errors in labels, particularly for instances where automated labelling may be prone to errors. • Statistical parity may 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. • Assess sampling bias using metrics such as the Wasserstein distance or the Kolmogorov-Smirnov statistic to quantify differences in data distributions [i.138]. • Consistency regarding sensitive entities can be measured by assessing the degree of agreement via Cohen's kappa between different labels or annotations for the same data point. Using a ground truth dataset allows a correlation comparison of consistent data points to determine the impact on bias [i.101]. • Introducing controlled anomalies and measuring the change in model can provide insight into how well the model handles unexpected or outlier data.	SL

Countermeasures	ML method
• Use stratified sampling to ensure proportional representation of different demographic groups in the training data. This helps prevent underrepresentation or overrepresentation of certain groups [i.118]. • Employ data augmentation techniques to artificially increase the diversity of the training data. This is especially useful when sensitive entities are underrepresented. • Identifying and removing sensitive characteristics which may foster bias. Due to changes in environmental conditions as well as tendencies in multifactorial development of forecasted parameters, supervised learning models may need to be retrained or updated to ensure unbiased results. • Include human annotators in the validation and correction of labels, especially for ambiguous or sensitive entities to control the quality of labels. • Missing values can be handled via imputation like mean/mode imputation, regression imputation, or using algorithms like K-Nearest Neighbours (KNN) [i.102]], [i.119]. • Outliers can be removed via removing, trimming and winsorising [i.120], [i.121]. • Analysing model performance before and after removing anomalies helps understand the impact on model correctness. • Resampling and reweighting to address class imbalance as well as bias by adjusting sample importance or modifying the dataset to ensure balanced and efficiency of model training.	SL

A.1.13 Cause: Insufficient quality of model capabilities

Insufficient quality of ML model capabilities can be reflected by deficiencies regarding correctness, robustness and bias, deteriorating the ML model's ability to e.g. classify, identify, compare, select, and/or create data.

Measures to detect	ML method
• Track performance metrics (e.g. confidence score, Intersection over Union, accuracy, precision, recall) over time and benchmark against standard datasets to identify any drops in correctness or performance inconsistencies. • Introduce noise or adversarial perturbations to input data to check if the model's predictions remain stable, helping detect issues related to model robustness. • Evaluate model performance on diverse datasets, including edge cases, outliers, and different demographics, to detect bias and generalization failures. • Conduct fairness audits using metrics like statistical parity and disparate impact to identify and mitigate any biases in the model's predictions across different groups. • Test the model on data from a different distribution than the training data to check for poor performance or failures in generalizing to unseen data.	SL/UL/RL

Countermeasures	ML method
• Implement bias and metrics (e.g. confidence score) constraints to ensure that outputs are appropriated. • Apply techniques (e.g. L2 regularization, dropout) during training to prevent overfitting to specific patterns, improving the model's ability to generalize and reducing its sensitivity to noise.	SL/UL/RL
• Train the model with adversarial examples to improve robustness against small input perturbations, ensuring the model remains stable and performs reliably under varied conditions. • Continuously retrain the model with fresh, diverse, and representative data to improve correctness and adapt to new trends, ensuring better generalization and reducing bias. • Use data augmentation techniques (e.g. adding noise, changing data distributions) to create more varied training data, enhancing the model's robustness and its ability to generalize across different scenarios.	SL

A.2 Causes related to exploiting the model behaviour

A.2.1 Cause: Evasion during inference/exploration/exploitation

Evasion refers to the act of bypassing or misleading a machine learning model's exploration/inference process, which can occur both intentionally (e.g. through adversarial attacks designed to deceive the model) or unintentionally (e.g. due to natural variations in input data or shifts in data distribution). In both cases, the result is a deviation from the expected output, potentially causing misclassifications or failures.

Measures to detect	ML method
• Compare the input data distribution against expected patterns. • Monitor for outputs that are unexpected or significantly deviate from the expected output. • Track confidence scores and focus on abnormally high or low values. • Employ adversarial detectors trained to distinguish between normal and adversarial examples. • Use techniques to trace predictions back to training data and identify if certain training examples are oversensitively influencing the output. • Simulate attacks like adversarial examples, poisoning, and evasion attacks to identify model vulnerabilities. • Use anomaly detection to identify unusual patterns or characteristics in inputs that deviate significantly from the training data distribution. • Compare the model's responses to similar or slightly perturbed inputs to detect inconsistencies indicative of evasion attempts. • Adversarial testing with malicious inputs to identify [i.122], [i.123] • induced misclassifications. • model misleading making inputs fall outside the model's training distribution. • inadequate regularization. • Over- and/or underfitting. • the model's robustness, correctness and unwanted bias. • Implement real-time monitoring of input data to detect deviations from expected patterns or distributions. • Monitor query logs for repetitive, systematic probing that may indicate an attempt to explore or manipulate the model's behaviour.	SL/RL/UL
• Analyse how sensitive the model is to changes in reward values, identifying areas where adjustments may lead to unwanted consequences [i.123]. • Use adversarial testing to identify potential exploits in the reward structure. • Monitor environmental interactions for abnormal changes in state or reward signals. • Use shadow models to compare expected outcomes against actual behaviour in the environment, flagging discrepancies.	RL

Countermeasures	ML method
• Build the model considering adversarial examples, which are purposely crafted to deceive the model. • Detect and remove adversarial noise. • Enforce boundaries for excessive load as well as numeric/categorical features to prevent extreme or out-of-range operation. • Application of explainable ML techniques to detect sensitivities that lead to undesirable results, e.g. biases and errors. • Implement a continuous monitoring system to detect deviations in model behaviour over time. • Retrain the model on recent and diverse data to adapt to threats. • Obfuscate the output to make it difficult for adversaries to directly access or infer gradient information from model queries. • Implement robust validation and sanitization (e.g. outlier detection) procedures to detect and filter out anomalous or malicious input data [i.125]. • Define and enforce strict input data format specifications to ensure that inputs adhere to expected patterns. • Anomaly detection systems to flag unusual input patterns that differ from the typical training data distribution. • Implement real-time monitoring systems to detect and block potential adversarial attempts using anomaly detection. • Implement data provenance systems that track the source and history of input data to ensure its integrity and identify potential poisoning or tampering attempts. • Use cryptographic hash functions or digital signatures to verify the authenticity and integrity of input data before it is used by the model. • Utilize adversarial example detection tools to filter out maliciously crafted inputs. • Use rate-limiting for unusual query activity to avoid potential exploration or inference manipulation attempts.	SL/RL/UL
• Create well-structured reward functions with constraints to prevent unintended optimization paths, and regularly test reward sensitivity to detect vulnerabilities. • Encrypt reward signals, validate them with cryptographic checks, and monitor for anomalies to prevent tampering. • Train RL agents with adversarially perturbed states and manipulated rewards to improve their robustness against input and reward manipulation. • Test the RL agent in controlled adversarial scenarios to identify and fix vulnerabilities in its reward structure and policy behaviour. • In RL systems, secure reward channels and state observations to prevent manipulation by adversaries. • Regularly test with simulated adversarial environments to identify and fix vulnerabilities in the exploration process.	RL

A.2.2 Cause: Tampered/poisoned training data.

Tampered or poisoned training data reflects modified or corrupted data introduced during the training phase to mislead the model, often resulting in biased or incorrect outputs. Such data poisoning can cause the model to learn false patterns, making it vulnerable to adversarial impacts or significantly degrading its performance.

Measures to detect	ML method
• Compare the hash values of the original training data with the current state of the dataset to identify any tampering. • Assess the statistical properties of the training data distribution. Any significant deviation may indicate tampering. • Evaluate the trained model's performance on a dataset known to be untampered. A drop in accuracy may indicate tampering [i.124].	SL

Countermeasures	ML method
• Regularly validate and audit training data to detect and prevent tampering. • Implement anomaly detection mechanisms to identify suspicious patterns. • Encrypt training data to protect it from unauthorized modifications. This ensures that even if data is accessed, it remains secure.	SL

Annex B:
Assessing the information security (potential risk sources for the criterion "security from vulnerabilities")

B.1 Cause: Lack of model integrity

A lack of model integrity in ML-based systems means that changes to the model or data are not properly tracked or secured, leading to inconsistencies and potential security risks. Without version control and integrity checks, models can be corrupted or tampered with, affecting their reliability and accuracy. Additionally, inadequate security measures may expose the model to unauthorized access or manipulation, further compromising its performance.

Measures to detect	ML method
• Implement version control for ML models to track changes. • Use hashing or checksums to verify the integrity of deployed models. • Overall system-related penetration testing to assess unauthorized accessibility of e.g. databases, model information as well as the backend in general. • Attempt to reconstruct or replicate the model architecture. • Embed a unique watermark into the model's output that can trace unauthorized replicas or stolen versions if e.g. inference/exploration/exploitation manipulation or reverse engineering occurs.	SL/UL/RL

Countermeasures	ML method
• Use encryption techniques to protect the model files stored on servers or in cloud environments, preventing unauthorized access or modifications [i.126]. • Enforce strict Role-Based Access Controls (RBAC) to limit who can view, modify, or deploy the model [i.129]. • Require Multi-Factor Authentication (MFA) for all access to model storage and deployment environments [i.130]. • Integrate security practices, including code reviews, dependency checks, and automated testing, to catch vulnerabilities early in the development process [i.131].	SL/UL/RL

B.2 Cause: Lack of data authenticity and integrity for training and output data

The absence of data authenticity and integrity in both training and output data increases the likelihood for data corruption, as there is no prove that it has remained unaltered since its creation or protected from unauthorized access.

Measures to detect	ML method
• Implement checks to verify the integrity of data used for training/generated by the model during deployment. • Penetration testing to assess unauthorized accessibility.	SL/UL/RL

Countermeasures	ML method
• Use digital signatures to sign data, ensuring its authenticity and integrity. • Include checksums and validation mechanisms in the output data processing pipeline to detect any data corruption or tampering [i.127].	SL/UL/RL

B.3 Cause: Lack of data encryption

Lack of encryption is caused when sensitive data, either during storage or transmission, is not encoded to prevent unauthorized access. This can occur due to negligence in implementing security protocols, inadequate security policies, or insufficient awareness of data protection requirements. Without encryption, data becomes vulnerable to breaches, hacking, or tampering, exposing it to potential misuse or theft.

Measures to detect	ML method
• Implement regular audits and network monitoring tools and scans of systems to identify any unencrypted data transmissions.	SL/UL/RL

Countermeasures	ML method
• Implement end-to-end encryption for data at rest, in transit, and during processing to protect it from unauthorized access [i.130]. • Use robust key management practices to secure encryption keys, ensuring that only authorized entities can decrypt the data [i.129]. • Use secure communication protocols such as SSL/TLS for encrypting data during transmission over networks [i.139].	SL/UL/RL

B.4 Cause: Insecure transmission channels

If transmission channels are insecure, data is sent over networks without proper security measures, making it vulnerable to interception or tampering. This can be caused by the use of outdated or unprotected communication methods, such as HTTP instead of HTTPS, or inadequate network security configurations.

Measures to detect	ML method
• Conduct vulnerability assessments and penetration testing to identify weak points in transmission channels.	SL/UL/RL

Countermeasures	ML method
• Use secure communication protocols (e.g. HTTPS) to protect data during transmission and prevent eavesdropping [i.140]. • Implement network-level encryption, such as Virtual Private Networks (VPNs), to secure data in transit. • Implement certificate pinning to validate the authenticity of SSL/TLS certificates during communication [i.139].	SL/UL/RL

B.5 Cause: Unauthorized access to deployed models

Unauthorized access to deployed models occurs when individuals or systems gain access to a model without proper authorization, often due to weak access controls or insufficient security measures. This can be caused by vulnerabilities in the model's deployment environment, such as inadequate authentication, misconfigured permissions, backdoors, supply chain attacks or exposed endpoints. Such unauthorized access can lead to tampering, theft of intellectual property, or misuse of the model for malicious purposes.

Measures to detect	ML method
• Regularly monitor access logs to deployed models and establish alerts for suspicious activities. • Test the model with both normal and adversarial inputs to identify triggers that cause consistent and intentional misclassifications to indicate adversary impact, e.g. backdoors. • Regularly audit the entire supply chain for unauthorized changes, focusing on dependencies, third-party libraries, and vendor-provided components.	SL/UL/RL

Countermeasures	ML method
• Enforce multi-factor authentication for accessing deployed models, adding an additional layer of security beyond just usernames and passwords [i.131]. • Use role-based access controls for finer-grained authorization [i.131]. • Secure APIs through which models are accessed. Use API keys, tokens, or other secure methods to control access and authenticate clients. • Assess the strength of user authentication mechanisms, including multi-factor authentication, to prevent unauthorized access.	SL/UL/RL

B.6 Cause: Insecure deployment environments

The deployment environment is insecure, when the infrastructure hosting the model lacks proper security measures, leaving it vulnerable to attacks. This can be caused by misconfigured servers, outdated software, weak access controls, or lack of network security protocols. As a result, the model and its associated data can be exposed to unauthorized access, tampering, or exploitation, compromising its integrity and security.

Measures to detect	ML method
• Conduct regular/continuous security audits and risk assessments of the deployment environment to identify and address vulnerabilities proactively. • Continuously monitor the deployment environment for unauthorized access, configuration changes, or anomalous activity that may indicate security vulnerabilities. • Use automated tools to scan for misconfigurations in servers, APIs, or cloud infrastructure (e.g. open ports, overly permissive access controls).	SL/UL/RL

Countermeasures	ML method
• Use containerization technologies like Docker to encapsulate models and their dependencies. This ensures consistent and secure deployment across various environments. • Implement strict authentication and authorization mechanisms (e.g. multi-factor authentication, role-based access control) and maintain detailed logs for forensic analysis. • Enforce security best practices, such as minimal privilege access, secure API gateways, encrypted communications (TLS/SSL), and disabling unused ports or services. • Ensure timely updates to the operating system, frameworks, and dependencies to mitigate known vulnerabilities.	SL/UL/RL

B.7 Cause: Traceability from inference to training data

Lack of traceability from inference to training data occurs when it is not possible to track or verify the relationship between the model's predictions (inferences) and the data used to train it. This can be caused by lack of explainability, poor data management practices, inadequate logging, or the absence of metadata to link data and model behaviour. Without traceability, it becomes difficult to audit, validate, or troubleshoot the model's performance on a fine-grained level, especially in the case of errors, bias, or unexpected outputs.

Measures to detect	ML method
• Conduct regular/continuous adversary simulations. • Test whether individual data points in the training dataset disproportionately influence the model's outputs, indicating potential traceability risks. • Log and search for suspicious or repetitive queries designed to infer details about the training data, such as systematic input-output probing.	SL

Countermeasures	ML method
• Implement logic filtering to block vulnerable requests. • Introduce noise into the model's outputs to obscure specific connections between training data and inference/exploration/exploitation output, reducing traceability. • Avoid exposing sensitive details like confidence scores or overly specific outputs that can aid attackers in reconstructing training data. • Limit the number and scope of queries from a single source (e.g. IP address) to hinder systematic probing for training data traceability.	SL

B.8 Cause: Reverse engineering

Reverse engineering in the context of machine learning involves an attacker probing a model by analysing its outputs to infer details about its structure, parameters, or training data. This process allows the attacker to replicate, exploit vulnerabilities, or craft more targeted attacks against the model.

Measures to detect	ML method
• Monitor for excessive, unusual, or systematic querying patterns, which may indicate an attempt to infer model behaviour or parameters through probing. Such can be designed to test slight variations of input to elicit differing outputs. • Log and analyse the model's in- and outputs to detect and trace unauthorized usage.	SL/UL/RL

Countermeasures	ML method
• Enforce query limits and quotas for users to prevent attackers from making a large number of queries within a short time frame. • Add controlled noise to the model's outputs to obscure the exact relationships in its predictions without significantly impacting usability, making reverse engineering more challenging. • Restrict output to generalized/abstract content rather than detailed or confidence-based responses, limiting the information attackers can extract from queries.	SL/UL/RL

B.9 Cause: DDoS attack

A Distributed Denial of Service (DDoS) attack involves overwhelming a model or system with a massive flood of traffic or requests, causing it to become unresponsive or unavailable. This disruption prevents legitimate users from accessing the service, often causing significant downtime and potential financial loss.

Measures to detect	ML method
• Deploy monitoring systems to track incoming traffic and identify unusual spikes or patterns that deviate from the norm, signalling a potential DDoS attack. • Implement rate-limiting mechanisms to restrict the number of requests from a single source.	SL/UL/RL

Countermeasures	ML method
• Validate the legitimacy of incoming requests to filter out malicious traffic. • Maintain and regularly update a blacklist of known malicious sources (e.g. IP addresses) to block traffic from suspicious sources proactively. • Limit the number of requests per source (e.g. IP address) within a specific time frame to reduce the impact of traffic floods. • Distribute incoming requests across multiple servers or instances to handle spikes in traffic effectively without overwhelming the system. • Deploy firewalls to filter and block malicious traffic based on pre-defined patterns or anomaly detection. • Use tools like CAPTCHA or bot detection mechanisms to differentiate between legitimate users and automated bots driving the attack.	SL/UL/RL

Annex C:
Assessing the safeguards against exploitation of ML model's inference/exploration/exploitation (Risk sources for the criterion "security from vulnerabilities")

C.1 Cause: Lack of robust input validation

Without proper input validation, malicious or malformed data can be processed by the ML model, leading to incorrect or unexpected outcomes during inference, exploration, or exploitation phases. This lack of safeguard enables the exploitation of model vulnerabilities, potentially causing incorrect predictions, model crashes, or adversarial attacks.

Measures to detect	ML method
• Verify if input validation rules are defined and actively applied to all incoming data, including types, ranges, and formats. • Manually test the model with malformed or unexpected inputs to see if they are rejected or processed improperly. • Verify that input data undergoes preprocessing and sanitization steps to prevent invalid or malicious data from entering the system.	SL/UL/RL

Countermeasures	ML method
• Ensure comprehensive validation of all input data, including checking for expected types, ranges, and formats. • Train a model used for pre-inference/exploration/exploitation to recognize and reject adversarial inputs by including adversarial examples during the training phase. • Clean inputs to ensure they conform to the expected specifications before processing by the model.	SL/UL/RL

C.2 Cause: Lack of rate limiting

When rate limiting is not enforced, an ML model becomes vulnerable to overload situations where excessive requests or inputs can overwhelm system resources. This can disrupt the system's ability to perform proper inference or exploration, leading to delayed responses or degraded performance, making the model susceptible to exploitation.

Measures to detect	ML method
• Verify if there is a cap on the number of requests a single user or IP can make within a given timeframe. • Analyse logs for high-frequency, repetitive access from the same source to see if the system is unprotected from rate-based attacks. • Look for instances of system performance degradation caused by high input volumes, which could indicate the absence of rate limiting.	SL/UL/RL

Countermeasures	ML method
• Use mechanisms to restrict the number of requests, for instance from a source (e.g. IP), within a set timeframe. • Add CAPTCHA challenges for users who make frequent requests within a short period. • Introduce throttling techniques to ensure that system resources are not overwhelmed by excessive requests.	SL/UL/RL

C.3 Lack of emergency measures for events of damage

Without emergency measures to detect and mitigate damage, an ML model is vulnerable to prolonged exposure to model failures or data corruption, which can go undetected until significant harm is done. This allows an attacker or unintended failure to degrade the system's inference accuracy or disrupt exploration/exploitation behaviours without immediate corrective action.

Measures to detect	ML method
• Verify that the system logs model inputs, outputs, and other relevant events (e.g. errors, prediction shifts). • Check if event logs for missing entries or periods are taken. • Check if the system includes mechanisms for detecting abnormal behaviour in real time (e.g. unexpected drops in accuracy or unusual model outputs).	SL/UL/RL

Countermeasures	ML method
• Implement real-time logging that captures all important events, especially model inputs and outputs, errors, and prediction statistics. • Set up automated alerts to notify administrators of any suspicious activity or anomalies in system behaviour. • Perform regular audits to ensure logging systems are capturing the necessary data for real-time analysis. • Implement emergency measures in case of identified anomalies (e.g. , system shutdown).	SL/UL/RL

C.4 Inadequate isolation of inference environments

Failure to properly isolate the inference environment from other system components exposes the ML model to potential cross-environment attacks or unauthorized data access. This weakens the security of the model's inference process, enabling attackers to influence the model's behaviour, manipulate predictions, or extract sensitive information during inference or exploration.

Measures to detect	ML method
• Ensure that the inference environment (e.g. the server or container where model inference occurs) is properly isolated from the rest of the infrastructure. • Review logs for unauthorized access attempts or unusual cross-environment communications, indicating potential failures in isolation. • Confirm that sensitive data and model operations are not sharing resources with other systems that could introduce vulnerabilities.	SL/UL/RL

Countermeasures	ML method
• Use secure containers or virtual environments to isolate the model's inference process from other system components. • Ensure that the inference environment has strict network boundaries, limiting access to only necessary systems. • Implement robust authentication and authorization to control who and what can interact with the inference environment.	SL/UL/RL

C.5 No obfuscation of model outputs

When model outputs are not obfuscated or sanitized, they may reveal sensitive details about the model's internal workings, training data, or biases. This lack of protection creates a vulnerability where attackers can exploit the model's behaviour or reverse-engineer its decision-making process, affecting its integrity during inference or exploitation [i.143].

Measures to detect	ML method
• Verify whether the model outputs raw or highly granular information that could reveal sensitive details about the training set or internal workings. • Monitor the output for any potentially revealing patterns, such as predictions that expose the model's internal structure or bias. • Review output data for exposure of sensitive training data that could be exploited or reverse-engineered.	SL/UL/RL

Countermeasures	ML method
• Use techniques like sanitization, normalization, rounding, adding noise, or limiting output precision to prevent sensitive data from being disclosed. • Apply differential privacy methods to ensure that outputs do not reveal information that could compromise privacy or model security [i.128]. • Limit the level of detail in model responses to reduce the risk of revealing sensitive insights.	SL/UL/RL

C.6 Lack of resource and load management

Without proper resource and load management, an ML model can become overwhelmed during high demand or peak usage times, leading to performance degradation or unhandled exceptions. This can negatively affect the model's ability to perform inference or exploration tasks, leaving it open to exploitation from users or attackers exploiting system slowdowns or failures.

Measures to detect	ML method
• Check for high levels of CPU, memory, or network usage during peak load times, indicating poor resource management. • Analyse logs for signs of performance bottlenecks, such as slow response times or unhandled exceptions, which may indicate insufficient load management. • Look for unusual spikes in incoming traffic that could overwhelm the system without adequate load balancing.	SL/UL/RL

Countermeasures	ML method
• Use load balancing techniques to evenly distribute traffic across available resources, preventing system overload. • Set up automatic scaling for infrastructure to handle fluctuations in demand without degrading performance. • Fine-tune resource allocation to ensure the system can handle peak loads effectively.	SL/UL/RL

C.7 Exploitable vulnerabilities in deployed environment (model containers or virtual machines)

If the deployed environment, such as containers or virtual machines, has unpatched vulnerabilities or misconfigurations, attackers can exploit these weaknesses to interfere with the model's functioning. This may result in unauthorized access, model manipulation, or disruption of inference and exploitation behaviours.

Measures to detect	ML method
• Regularly scan containers or virtual machines for known vulnerabilities and misconfigurations that could be exploited by attackers. • Ensure that the operating system and dependencies within containers or VMs are up-to-date with the latest security patches. • Review access logs for signs of unauthorized access or suspicious activities targeting the containerized environment.	SL/UL/RL

Countermeasures	ML method
• Regularly update containers, VMs, and underlying software to mitigate known vulnerabilities. • Deploy the model in secure, hardened containers that provide built-in security features and automated vulnerability assessments. • Limit access to containerized environments.	SL/UL/RL

C.8 No adversarial example detection

In the absence of adversarial example detection, the ML model becomes vulnerable to adversarial inputs designed to deceive or mislead the model, affecting its decision-making or predictions. This can lead to manipulation of the model's inference or exploitation behaviours, allowing attackers to induce incorrect outputs without detection [i.88].

Measures to detect	ML method
• Verify whether the model has dedicated mechanisms in place for detecting adversarial inputs, such as specialized detection algorithms or defensive models. • Test the model with known adversarial examples and check whether the system can identify and reject them. • Review the model's performance in scenarios where adversarial examples are injected, and measure if it maintains robustness or fails unexpectedly.	SL/UL/RL

Countermeasures	ML method
• Use specialized adversarial example detection tools that can flag and reject malicious inputs before they reach the model. • Apply robust optimization techniques to train a model to handle potential adversarial perturbations for pre-inference/exploration/exploitation.	SL/UL/RL

C.9 Behavioural consistency monitoring

Without monitoring for behavioural consistency, sudden shifts or manipulations in the model's outputs may go unnoticed, leaving the system vulnerable to exploitation. Anomalies or adversarial interference in inference or exploration phases can be undetected, compromising the model's reliability and security.

Measures to detect	ML method
• Monitor model outputs to ensure consistent behaviour under similar conditions and inputs to identify exploitable/exploited locations. • Review logs for sudden shifts in output or prediction patterns that could suggest manipulation or adversarial influence. • Analyse the model's responses across different environments to ensure consistency in predictions and actions.	SL/UL/RL

Countermeasures	ML method
• Implement continuous behavioural consistency checks to compare current outputs with expected predictions based on past data. • Use anomaly detection systems to identify deviations from normal behaviour, which could signal manipulation or adversarial activity. • Establish baseline models to compare new outputs against, ensuring that any unexpected behaviour is flagged and addressed.	SL/UL/RL

C.10 Backdoor injection during training

If backdoors are injected into the training process, the ML model may behave as expected under normal conditions but exhibit harmful behaviour when specific triggers are encountered. This creates vulnerabilities in the model's inference and exploration behaviours, allowing for targeted exploitation when the backdoor is triggered [i.60].

Measures to detect	ML method
• Review training data to check for unusual patterns or outliers that could indicate the presence of backdoor attacks. • Monitor model behaviour during testing for unexpected or targeted predictions when certain trigger inputs are applied. • Investigate any discrepancies in model performance across different datasets that could indicate the presence of hidden backdoors.	SL

Countermeasures	ML method
• Use robust data sanitization techniques to remove potentially malicious data during the training process. • Perform rigorous testing using adversarial and backdoor-specific testing methods to detect hidden triggers in model behaviour. • Implement model verification methods such as differential testing to compare the model's responses under controlled conditions.	SL

C.11 Model corruption during deployment

Corruption of the model during deployment, whether due to malicious tampering or technical failures, can alter the model's behaviour and decision-making processes. This degradation can interfere with inference accuracy and exploration, opening the door for exploitation by attackers who can manipulate the model's performance.

Measures to detect	ML method
• Regularly verify the integrity of the model in the deployment environment, ensuring no unauthorized modifications have been made. • Monitor model performance over time to detect any sudden degradation or performance shifts that could indicate corruption. • Use version control systems to track and validate model updates, ensuring that no unexpected changes have occurred during deployment.	SL/UL/RL

Countermeasures	ML method
• Use secure deployment pipelines to ensure that models cannot be tampered during updates. • Implement model integrity checks, including cryptographic signatures or hash checks, to verify that the deployed model matches the intended version. • Conduct regular audits of the deployed model and compare its performance with baseline expectations.	SL/UL/RL

C.12 Exploratory Data Manipulation

Exploratory data manipulation involves intentionally altering data inputs to detect weaknesses and identify patterns that can be used to circumvent the ML model's inference/exploration/exploitation protections.

Measures to detect	ML method
• Audit the data pipeline to identify potential points where adversaries could manipulate or alter training or inference data. • Look for unusual patterns in data distribution that may suggest an attacker has influenced the model by altering the input data. • Analyse the model's predictions over time to identify any systematic biases or errors that could arise from manipulated data.	SL/UL/RL

Countermeasures	ML method
• Implement data integrity checks and validation at all stages of the data pipeline to prevent manipulation. • Use anomaly detection systems to identify outliers or unusual data patterns that could indicate tampering. • Introduce robust data preprocessing techniques that can identify and neutralize data manipulation before it reaches the model.	SL/UL/RL

Annex D:
Questionnaire for explaining ML-based systems

The following questionnaire has been designed to serve as a guideline for understanding and explaining machine ML-based systems. This set of questions aims to provide clarity and insights into the intricacies of ML models, aiding in the development of clear and accessible explanations. The questions are structured to cover various aspects, allowing stakeholders to assess and articulate ML-related characteristics associated with the ML model's operation, impact, risks, conformity with standards as well as legislations, sensitivity considerations, model relevance, uncertainty quantification, statistical distribution, system architecture, deployment constraints, and adaptability.

Characteristic to be assessed	Question
1. The ML-based system's impact	
• Impact on interacting components	How does the ML-based system's behaviour impact other components in the system?
• Impact on overall system	How does the ML-based system's behaviour impact the overall system's functionality and performance?
2. Risk Characteristics	
• Damage potential on data	What are the potential risks associated with the ML model's impact on data?
• Damage potential on finance	How are the financial risks associated with the ML model's predictions or decisions communicated?
• Damage potential on human behaviour	Are there any risks related to behavioural aspects related to humans, such as nudging, sludging, or context manipulation?
• Damage potential on physical and mental well-being	How are potential risks to the physical and mental well-being of users addressed?
3. Conformity assessment procedures	How does the ML-based system demonstrate compliance with legislations, standardization documents as well as ethical frameworks?
4. Sensitive entities involved in model development	Is there transparency about the involvement of sensitive entities as well as biases in the ML model's development, e.g. training data?
5. Rationale of using the model (Annex E)	What is achieved by model training/inference/exploration/exploitation?
6. Model development procedures (Annex E)	Which steps were taken for developing the model?
7. Processes during operation (Annex E)	Which processes are carried out during the model's operation?
8. Model relevance	
• Application context	How is the MI model relevant to the specific application context, e.g. does it align with the intended purpose and domain?
• Intended use	What is the intended use of the model, especially specifying the scenarios and tasks for which it is designed?
• Implemented methods and capabilities	How are the implemented methods and capabilities of the ML model communicated, e.g. in product accompanying documents?
• Tasks to be realized	What tasks is the ML-based system intended to accomplish, and how are its capabilities communicated to stakeholders?
• Quality and performance characteristics	Which quality and performance characteristics of the ML-based system are identifiable?
9. Uncertainty quantification of sensitive entities in inference/exploration/ exploitation results	How can the uncertainty associated with sensitive entities in the model's inference/exploration/exploitation results be quantified?
10. Statistical distribution of sensitive entities in training data set	Is the statistical distribution of sensitive entities within the training dataset disclosed?
11. Overall system around the integrated model	How is the ML-based system integrated into the overall supersystem?
12. Constraints of model deployment during the intended use	Which constraints/limitations exist regarding the ML model's deployment during its intended use?
13. Interaction with overall system components	How does the ML-based system interact with other components in the overall supersystem?
14. ML-based system's adaptability	Which characteristics of the ML-based system change over time?

Annex E:
ML models: Explaining rationale, development and operation

The following table shows exemplarily how the rationale, development and operation of ML models can be explained, [i.14], [i.94].

ML model: rationale, building & operation	Explanation
Examples for rationales:	**Model training:** • The model learns patterns and relationships between input features and target labels from the labelled training data. **Model inference (SL, UL):** • Creation of predictions on unseen data. • Identification of inherent patterns, structures, or relationships within the input data with/without guidance from labelled examples. • Grouping of similar instances to clusters. • Reducing the dimensionality of inferred data. **Model exploration (RL):** • Learning by interacting with the environment and receiving feedback in the form of rewards or penalties (RL). • Making a sequence of decisions over time to maximize cumulative rewards (RL).
Model development procedure:	**Algorithm selection:** • Selection of a method suitable for a specific task (e.g. decision trees, self--organizing maps, k-means, q-learning, neural networks...). **Data preprocessing:** • Handling of missing values, scale features, and encode categorical variables if needed. **Hyperparameter tuning:** • Optimization of model parameters for better performance. **Model assessment:** • Assessment of model performance on validation data. **Supervised Learning / Unsupervised Learning:** • **In case of model training:** Utilization of labelled examples, consisting of input features and corresponding target labels, to train the model by adjusting parameters to maximize the quality of predictions, e.g. the confidence score. • Implementation of suitable functions enabling to identify specific features during inference. **Reinforcement Learning:** • **In case of model training:** Interaction with the environment, reception of rewards, and update of the policy or value function through iterative learning. • Definition of states, actions, and rewards in the environment. • Initialization of the policy or value function.
Processes during operation:	**Inference:** • Identification of relevant features from unseen input data to make predictions. **Exploration:** • State representation: Representation of current states in the environment. • Action selection: Selection of actions based on the current state, informed by learned policies or value functions. • Reward feedback: Reception of rewards or penalties based on actions taken. • Policy/Value update: Adjustment of the policy or value function based on received rewards to improve decision-making over time.

History

Document history		
V1.1.1	February 2025	Publication