How to Safeguard AI

Ina Schieferdecker/Jürgen Großmann/Martin A. Schneider

1. Introduction

Artificial intelligence is a discipline within computer science that deals with the development of software-based systems that provide functions which require the execution of what is typically called (human) intelligence. However, since there is no widely accepted definition of human intelligence, there is also no widely accepted for artificial intelligence, sometimes also called machine intelligence (Legg, 2007). AI uses methods and tools from logic, probability theory, and continuous mathematics in order to provide perception, reasoning, learning, and action via software-based systems (Russell, 2016). And it provides already numerous practical applications in transportation, energy supply, health services, finance and banking as well as law and regulation: "AI technologies already pervade our lives. As they become a central force in society, the field is shifting from simply building systems that are intelligent to building intelligent systems that are human-aware and trustworthy." (Stone, 2016)

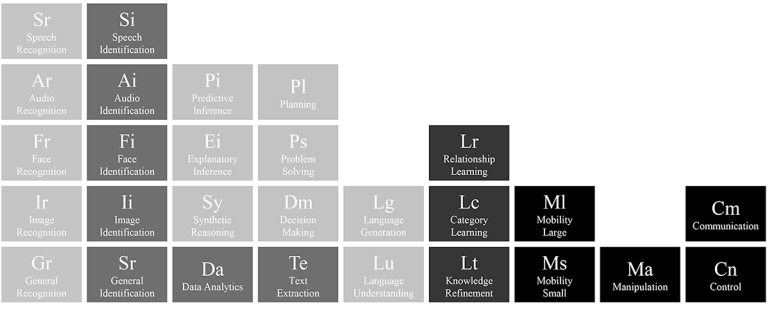

Fig. 1: Functional components in AI by Hammond (2016): Recognition of speech (Sr), audio (Ar), face (Fr) and image (Ir) and general recognition (Gr), Identification of speech (Si), audio (Ai), face (Fi) and image (Ii) and general identification (Gi); Data analytics (Da) and Text extraction (Te); Predictive inference (Pi), Planning (Pl), Explanatory inference (Ei), Problem solving (Ps), Synthetic reasoning (Sr), and Decision making

(Dm); Language generation (Lg) and understanding (Lu); Relationship learning (Rl), Category learning (Cl) and Knowledge refinement (Kr); Mobility at large (Ml) and at small (Ms); Manipulation (Ma), Communication (Cm) and Control (Cn), which can be used standalone or in combination e.g. to predict future events by recognizing sounds of technical systems and/or identifying images representing system states and/or correlating data and recognizing specific facts.

Technologies that are used to build AI by machine learning (in short ML), which is about improving problem solving accuracy or efficiency by learning to do something better, are numerous. Machine learning can e.g. be grouped along the learning type into methods for supervised, unsupervised or semi-supervised learning or along the knowledge extraction by symbolic computation or sub-symbolic processing. They can also be grouped along the principal approach, e.g. into regression, instance-based, regularization, decision tree, Bayesian, clustering, neural network, deep learning, and quite many other algorithms. Based on these, likewise numerous AI applications can be developed. Hammond (2016) presented a first taxonomy of AI functional components (Fig. 1). No matter which functional components are being used, AI-based systems are realized by use of software or also by use of sensors and actuators for the interconnection with the environment (Fig. 2). The software uses data which are interpreted by algorithms in order to provide automatisms for parts of or for entire processes in technical systems like in car engine control or in socio-technical systems like in autonomous driving.

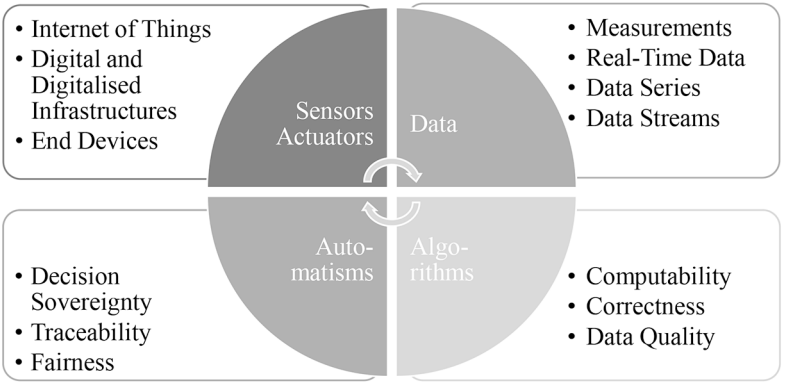

Fig. 2: Elements of software-based systems (WBGU, 2019). Sensors are part of the Internet of Things and generate different kinds of data such as measurements, series of measurements or data streams. Algorithms use these data in their computations or as training data. The algorithms are constrained by complexity, computability, and performance limits and possibly by the (in-)correctness of the implemented computation logic and by the (un-)biased (training) data. In result, software-based systems offer

automatisms for which it is essential to agree (and assure) decision sovereignty, traceability and fairness. Any decision in respect to the environment can finally be fed via software (into the cyberspace) and via actuators (into the environment).

2. Software Verification and Validation

Since any AI is also a software-based system, it is to be seen to which extent AI can be verified and validated with the established verification and validation (in short V&V) methods for software in general. V&V methods for software were revealed already with the software crisis back in 1968 (Wirth, 2008), when the term software engineering was coined. It pointed at the difficulties to design and develop useful and trustworthy software with the given resources and within the given time: "The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! …(A)s long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." (Dijkstra, 1972). And the newly coined term pointed at the necessity to develop practical and scalable engineering methods for software development. Since then, constructive and analytic methods for software quality engineering have been developed. They include methods for software engineering processes, software engineering tools and for software as such. A rough overview on these methods is given in Fig. 3.

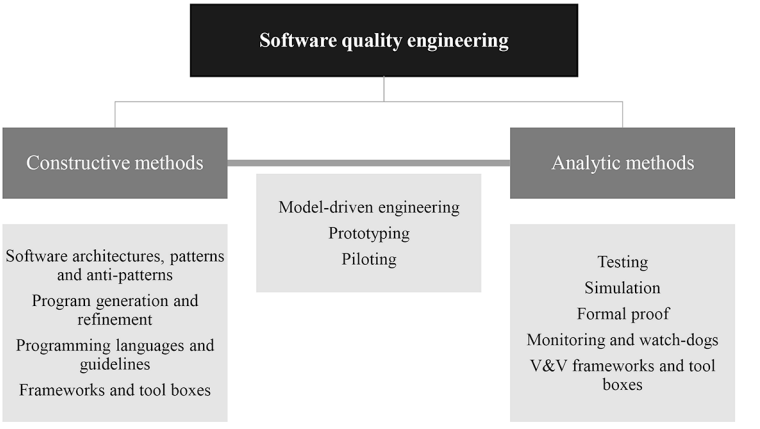

Fig. 3: Overview on software quality engineering methods. Software quality begins with the software design that is represented by software architectures which can make use of software patterns. Programs can be (partially) generated from these software designs and/or refined. The programs use typically high-level programming languages

which offer guidelines for best practice programming and which are supported by programming frameworks and tools. The achieved software quality is typically tested, checked by simulation or proven formally. The running software can be monitored and watch-dogs can check for constraint violations at run-time. All these analytic methods can also be automated by V&V frameworks and tools. Three specific (sets) of methods can be used both constructively and analytically: that is the use of model in software engineering, the early prototyping of software (or of V&V software) and the piloting of software (or of V&V solutions).

The software (program or code) tells the computer what to do, *"but that may be much different from what you had in mind".* (Joseph Weizenbaum, Computer Scientist, 1923-2008). However, by the systematic use of software quality engineering methods, software can be developed such that it is safe, secure, and trustworthy and that it can analyze and compute more data than any person and can do this more reliably.

Numerous international software engineering standards put the ground for software quality such as ISO/IEC 25010 (ISO, 2011) for software quality requirements and evaluation (SQuaRE) and software quality models. It argues about quality in use, external quality and internal quality of software and differentiates between functional suitability, reliability, usability, security, compatibility, portability, maintainability and performance/efficiency.

While these are all important software quality aspects that evolved over decades, interestingly, new aspects arise for AI in their use within socio-technical systems. Apparently,

- understandability, i.e. users and operators can get to know the features and services of the systems,
- interpretability, i.e. users and concerned people have access to clarifications of outcomes and their potential impacts,
- traceability, i.e. users and concerned people have access to more detailed analysis of outcomes in relation to a given situation/problem statement,
- explainability, i.e. users and concerned people receive descriptions, reasoning and justifications on the outcomes, as well as
- fairness, i.e. concerned people are treated the same wrt. commonly agreed rules for treatment, gain much more momentum.

3. AI Verification and Validation

Indeed, AI requires to quite some extent additional methods and tools for V&V (Van Wesel, 2017) since well-established testing technologies are short in V&V of AI. This is not only true because of the additional socio-technical quality aspects (see above), but also due to the different nature of logic-based software (most of the software in general so far and some of AI) and statistics-based software (most of AI, in particular in machine learning). Testing has limitations with respect to the dynamics of ML, the sheer size of the problem domain and the underlying oracle problem (Xie, 2011).

In addition, most of the AI is controlled by data. In this sense, a neural network is a generic function approximator whose structure reflects the actual functionality only to a very small extent. Hence, source code-oriented V&V techniques such as static analysis or white-box tests are only of limited use in this context. On the other hand, the trustworthiness and quality of the data becomes a central issue for the overall quality of the systems.

However, since systematic dynamic testing of software is the best-known and most effective V&V method, it will most probably also form the main basis for testing ML. In recent decades, research has developed industrial-grade techniques for increasing the quality, efficiency and reliability of testing. This includes in particular, automation strategies for dynamic testing such as automating test executions with test technologies like TTCN-3 (Testing and Test Control Notation [Grabowski, 2003]), for model-based testing to automate the generation of tests (MBT [Utting, 2012]), as well as the use of search and optimization algorithms for automated test selection and test suite reduction (Harman, 2015). Moreover, the combination of dynamic testing with verification approaches like source code analysis, model checking and symbolic execution allows for improvements in testing, that combines the rigor of verification processes with the scalability of dynamic testing (Godefroid, 2018). These techniques are applied to testing for functional as well as extra-functional properties like performance or security (Schieferdecker, 2012). Finally, the close integration of testing with system development processes and risk management (Felderer, 2014) improved the efficiency and transparency of testing so that testing has matured as one of the most important software quality measures in industry. Still, test automation as well as the use of models in testing are still underexplored: although a strong test automation is required, less than 14% of software testing professionals say that they use MBT (Binder, 2015). The potential of risk-based testing to steer test processes based on uncertainties has been shown especially in the area of critical system in terms of security and safety, which will likewise be applicable to AI (Erdogan, 2014).

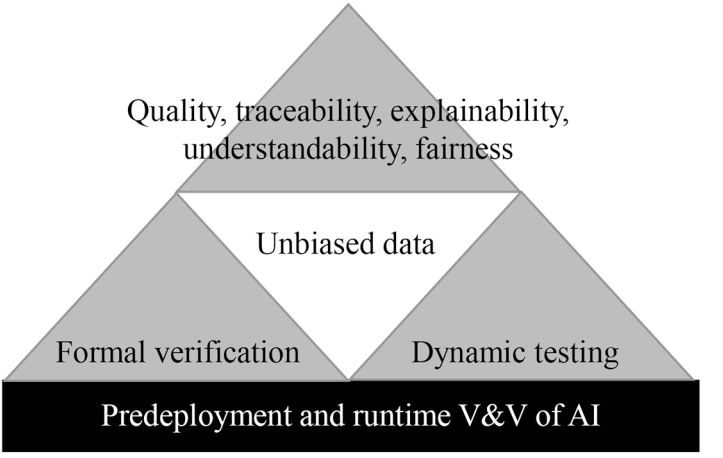

Fig. 4: The AI V&V pyramid. AI-based systems are to be verified and validated both in predeployment phases and at runtime. A combination of V&V methods from formal verification and dynamic testing is recommended, in particular for safety- and security critical AI-based systems. V&V will help to assure both quality and explainability requirements as well as enable the justification of bias in the (training) data used in AI.

Research on dedicated methods for verification and validation of ML is still at its beginning. Even so, testing is already part of the overall training set-up in ML, most testing is done to achieve more accurate models with respect to the initial training objectives. In supervised learning for example, test and validation data sets are used to provide evaluation of the ML model. Validation data sets are typically used during training to fine-tune the model parameters while test data sets are used on the final model to measure generalization errors. However, since individual test sets only provide a single evaluation of the model and have limited ability to characterize the uncertainty in the results, more advanced statistical testing approaches like cross-validation are used for model selection.

Ghosh et al (2016) combine ML and model checking in such a way that if the desired logical properties are not satisfied by a trained model, the model ('model repair') or the data from which the model is learned is modified systematically ('data repair'). Fulton and Platzer (2018) propose to combine formal verification with verified runtime monitoring so that safe learning can be guaranteed. The approach intervenes in the learning process whenever safety properties are violated and guides the learning process so that the result is compliant with the verification model. Approaches like DeepXplore (Pei, 2017), DLFuzz (Guo, 2018) and TensorFuzz (Odena, 2018) provide metrics for the quantification of neural coverage and simplify test automation. DeepTest (Tian, 2018) enables systematic testing of

neural networks under realistically changing environmental conditions especially for use in the automotive domain.

One of the socio-technical limitations of ML is the lack of transparency, i.e. its black box-approach. In order to address it, different approaches have been proposed such as

- model interpretation for image classifications, e.g. by understanding the activation maximization with saliency maps (Simonyan, 2013),
- model explanation by sensitivity analysis and local explanation vectors to provide reasons for the decisions of any classification method (Baehrens, 2010),
- model decomposition for interpreting generic multilayer neural networks by decomposing the network classification decision into contributions of its input elements (Montavon, 2017),
- extraction of decision trees from input data generated from trained neuronal networks (Krishnan, 1999),
- relevance propagation by pixel-wise decomposition of non-linear classifiers (Bach, 2015), and
- deconvolution methods to give insight into the function of intermediate feature layers and the operation of classifiers (Zeiler, 2014).

Another well-established way is to use test scenarios, i.e. test cases and their test data, for explaining ML decisions. The other socio-technical limitation of ML is the potential lack of fairness, i.e. the potential bias. Here, systematic generation of (training) data that cover well required categories and properties as known from test data generation is of help (Nguyen, 2016).

The ability to effectively test AI will be fundamental for the acceptance in broad scale and central for safety-critical areas like transportation and automotive, healthcare, or industrial automation. The provisioning of test technologies, tools, test scenarios with test cases and test data for AI will not only be a solid basis for V&V but also help in explaining AI and making them more transparent and unbiased. They can also be used to ensure safety and security of AI during runtime.

And last but not least, the tools for safeguarding AI contribute also to the democratization of AI: They are the basis for confirming or witnessing outcomes whenever AI-based systems are to be accounted. They can also become a digital common for the comparison and benchmarking of AI-based systems and by that contribute to a shared knowledge basis of AI.

Acknowledgment

This article resulted from several discussions with the research group on the criticality of AI-based systems lead by Diana Serbanescu, with the German Advisory Council on Global Change lead by Dirk Messner and Sabine Schlacke, with the participants at the conference "The Democratization of AI. Net Politics in the Era of Learning Algorithms" in Bochum, September 2018 led by Andreas Sudmann and with the members of the ITEA3 project "Industrial-grade Verification and Validation of Evolving Systems (IVVES) led by Jürgen Großmann.

References

Bach, Sebastian, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek (2015): "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation", PloS one 10, no. 7, e0130140.

Baehrens, David, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller (2010): "How to Explain Individual Classification Decisions", Journal of Machine Learning Research 11, no. Jun, 1803-31.

Binder, Robert V, Bruno Legeard, and Anne Kramer (2015): "Model-Based Testing: Where Does It Stand?", Communications of the ACM 58, no. 2, 52-56.

Dijkstra, Edsger W. (1972): "The Humble Programmer", Commun. ACM 15, no. 10, 859-66.

Erdogan, Gencer, Yan Li, Ragnhild Kobro Runde, Fredrik Seehusen, and Ketil Stølen (2014): "Approaches for the Combined Use of Risk Analysis and Testing: A Systematic Literature Review", International Journal on Software Tools for Technology Transfer 16, no. 5, 627-42.

Felderer, Michael, and Ina Schieferdecker (2014): "A Taxonomy of Risk-Based Testing", International Journal on Software Tools for Technology Transfer 16, no. 5, 559-68.

Fulton, Nathan, and André Platzer (2018): "Safe Reinforcement Learning Via Formal Methods: Toward Safe Control through Proof and Learning", Paper presented at the Thirty-Second AAAI Conference on Artificial Intelligence.

Ghosh, Shalini, Patrick Lincoln, Ashish Tiwari, Xiaojin Zhu, and Wisc Edu (2016): "Trusted Machine Learning for Probabilistic Models", Paper presented at the ICML Workshop on Reliable Machine Learning in the Wild.

Godefroid, Patrice, and Koushik Sen (2018): "Combining Model Checking and Testing", In Handbook of Model Checking, 613-49: Springer.

Grabowski, Jens, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, and Colin Willcock (2003): "An Introduction to the Testing and Test Control Notation (Ttcn-3)", Computer Networks 42, no. 3, 375-403.

Guo, Jianmin, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun (2018): "Dlfuzz: Differential Fuzzing Testing of Deep Learning Systems", Paper presented at the Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

Hammond, Kris (2016): "The Periodic Table of AI", https://www.datasciencecentral.com/profiles/blogs/the-periodic-table-of-ai. Accessed 11 July 2019.

Harman, Mark, Yue Jia, and Yuanyuan Zhang (2015): "Achievements, Open Problems and Challenges for Search Based Software Testing", Paper presented at the IEEE 8th International Conference on Software Testing, Verification and Validation (ICST).

ISO/IEC 25010 (2011): Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (Square)-System and Software Quality Models. International Organization for Standardization.

Krishnan, R, G Sivakumar, and P Bhattacharya (1999): "Extracting Decision Trees from Trained Neural Networks", Pattern recognition 32, no. 12.

Legg, Shane, and Marcus Hutter (2007): "Universal Intelligence: A Definition of Machine Intelligence", Minds and machines 17, no. 4, 391-444.

Montavon, Grégoire, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller (2017): "Explaining Nonlinear Classification Decisions with Deep Taylor Decomposition", Pattern Recognition 65, 211-22.

Nguyen, Anh, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune (2016): "Synthesizing the Preferred Inputs for Neurons in Neural Networks Via Deep Generator Networks", Paper presented at the Advances in Neural Information Processing Systems, 3387-3395.

Odena, Augustus, and Ian Goodfellow (2018): "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing", arXiv preprint arXiv:1807.10875.

Pei, Kexin, Yinzhi Cao, Junfeng Yang, and Suman Jana (2017): "DeepXplore: Automated Whitebox Testing of Deep Learning Systems", Paper presented at the proceedings of the 26th Symposium on Operating Systems Principles, 1-18, ACM.

Russell, Stuart J, and Peter Norvig (2016): Artificial Intelligence: A Modern Approach. Malaysia; Pearson Education Limited.

Schieferdecker, Ina, Juergen Grossmann, and Martin Schneider (2012): "Model-Based Security Testing", arXiv preprint arXiv:1202.6118.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman (2013): "Deep inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", arXiv preprint arXiv:1312.6034.

Stone, Peter, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Sarit Kraus, Kevin Leyton-Brown, David Parkes, William Press, AnnaLee Saxenian, Julie Shah, Milind Tambe, and Astro Teller (2016): "Artificial Intelligence and Life in 2030. One Hundred Year Study on Artificial Intelligence. Report of the 2015 Study Panel", 52. Stanford, CA: Stanford University.

Tian, Yuchi, Kexin Pei, Suman Jana, and Baishakhi Ray (2018): "DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars", Paper presented at the Proceedings of the 40th International Conference on Software Engineering, 303-314. ACM.

Utting, Mark, Alexander Pretschner, and Bruno Legeard. "A Taxonomy of Model-Based Testing Approaches", Software Testing, Verification and Reliability 22, no. 5 (2012): 297-312.

Van Wesel, Perry, and Alwyn E Goodloe (2017): "Challenges in the Verification of Reinforcement Learning Algorithms", NASA/TM–2017-219628.

WBGU (2019): Towards Our Common Digital Future. Flagship Report. Berlin: German Advisory Council on Global Change.

Wirth, Niklaus. "A Brief History of Software Engineering", IEEE Annals of the History of Computing 30, no. 3 (2008): 32-39.

Xie, Xiaoyuan, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen (2011): "Testing and Validating Machine Learning Classifiers by Metamorphic Testing", Journal of Systems and Software 84, no. 4, 544-58.

Zeiler, Matthew D, and Rob Fergus (2014): "Visualizing and Understanding Convolutional Networks", Paper presented at the European conference on Computer Vision, 818-833, Springer.