All Agents Created Equal?

The Law's Technical Neutrality on AI Knowledge Representation

by **Philipp Lerch***

Abstract: The term "Artificial Intelligence" comprises different approaches. They can be roughly divided into rule-based approaches and approximative machine learning. The author discusses the legal implications of this technological choice on the background of Product Liability law. It stands to reason that using rule-based approaches may be prone to stricter safety standards than approximative implementations.

Keywords: Product Liability; Product Security; Artifical Intelligence

Recommended citation: Philipp Lerch, All Agents Created Equal? The Law's Technical Neutrality on AI Knowledge Representation, 14 (2023) JIPITEC 108 para 1.

A. Introduction

1 A recent EU Commission's proposal aims at amending the legal framework on Product Liability with specific adaptations for products employing Artificial Intelligence technologies.[1] It is part of a major strategy of the European Union embracing the fields of Product Security, Technology Regulation and Contractual Liability, *inter alia*. The proposed directive adapts "non-contractual fault-based civil liability rules to artificial intelligence".[2] The most eye-catching though unspectacular novelty is—not surprisingly—the codification of the widely accepted notion that software is indeed a product (Article 4, para 1 of the Directive). The changes made appear to be rather subtle (which is, simply put, a smart decision disregarding those hyped voices who cannot wait to introduce AI Law early enough as a fourth major area of law). Interestingly the two major concerns of what forms a *defect* (as the most central term of Product Liability Law), and what justifies *exculpation* are not extended by a fundamentally new approach. Article 6, para 1 of the Directive amends certain circumstances to take into account when a defect is being ascertained:

2 "The effect on the product of any ability to continue to learn after deployment" (lit. c) refers to what is known as "development risks" of AI systems in the debate. The effect on the product of other products that can reasonably be expected to be used together with the product" (lit. d) can be described as interoperability which has already been set for the

* Philipp Lerch, Formerly Institute for Legal Informatics, Saarland University.

1 COM(2022) 495 - Proposal for a directive of the European Parliament and of the Council on liability for defective products.

2 COM(2022) 495, Explanatory Memorandum 1.2.

term of contractual defect.[3] With lit. e the aspect is taken into account that products may be kept under control of the manufacturer via network connection.[4] Lit f) and g) state that product safety requirements including cybersecurity, as well as "specific expectations of the end-users for whom the product is intended" are to be taken into account which is nothing revolutionarily new to the doctrine of Product Liability.

3 On the *exculpation* side the relevant Article 10, para 1 provides even less deviations from the current law. The exemption ground of lit. e) is still central, which allows exculpation if the defect could not have been discovered due to the objective state of scientific and technical knowledge at the time when the product was put on the market.

4 One problem identified in the field of AI law is whether self-learning systems, whose behaviours change over time, are subject to liability also for the adaptions that occur after the user has put the product into operation.[5] The novel directive surely aims at solving this issue. However, it assumes that most systems' algorithms do not evolve in the hand of the user. In principle, a computer software can (somewhat) solve *any* problem *either* by coding it to explicitly implement algorithms *or* by "training" how to solve it. This touches even more fundamental issues that are not tackled by the Directive at all. It goes to the heart of a Product Liability legal regime and touches specifically technical concerns: What *constitutes* a defect? Was it *avoidable*? And if it was, was it also *discoverable*?

5 A manufacturer may make use of machine learning techniques instead of coding the system's behaviour explicitly. The most illustrating examples for this can be found in the field of autonomous vehicles. There is ongoing research regarding so-called "end-to-end" approaches for autonomous vehicle control.[6] Instead of classical modular development of the vehicle, a single machine learning model is trained on the entire driving functionality like steering, object and lane detection, path planning, and control.[7] In such a framework information about the outer world ("knowledge"), particularly the way a vehicle ought to behave, is not being provided explicitly to the vehicle. Instead, it is being implicitly induced by the training data, that could be obtained by a human driver in operation.

6 The classical way autonomous vehicles are being constructed is different: expert and world knowledge, particularly traffic rules are being explicitly coded.[8] They serve as explicit constraints over other modules that make use of machine learning algorithms.

7 I will call the latter approach "explicit rule based". World knowledge leading to an agent's behaviour is being explicitly represented and the system operates directly on it. The former approach is the "implicit" machine learning approach. The agent's behaviour results from the induction of rules (implicitly represented in the system) from a given set of data. The choice of whether to use either of the methods also affects the widely-known postulate of transparency (problem of opacity): many machine learning techniques suffer from poor interpretability, known as the *black box* problem.

8 Unfortunately, there has not been active research on the legal consequences of this choice. Is the law technically neutral on this question? Another EU proposal, the famous AI Act[9], has been overtly called "technically neutral".[10] Technical neutrality means that the law is not per se preferring one technical approach to another in a specific domain, neither it is imposing a specific regime on any technical solution. Recent legislation is being called "technically neutral" as the regulators may have explicitly enumerated the (almost) entire set of

3 Directive (EU) 2019/771 of the European Parliament and of the Council of 20 May 2019 on certain aspects concerning contracts for the sale of goods, amending Regulation (EU) 2017/2394 and Directive 2009/22/EC, and repealing Directive 1999/44/EC, OJ L 166 0f 22 May 2019 ("SGD"), Art 2(5)(b).

4 "The moment in time when the product was placed on the market or put into service or, where the manufacturer retains control over the product after that."

5 Ebers, „Autonomes Fahren: Produkt- und Produzentenhaftung", in: Oppermann and Stender-Vorwachs, *Autonomes Fahren. Rechtsfolgen, Rechtsprobleme, technische Grundlagen*, p 34 ff.

6 For instance, see Rausch et al, "Learning a Deep Neural Net Policy for End-to-End Control of Autonomous Vehicles", 2017 American Control Conference (ACC) (24-26 May 2017).

7 Rausch et al, "Learning a Deep Neural Net Policy for End-to-End Control of Autonomous Vehicles", 2017 American Control Conference (ACC) (24-26 May 2017).

8 See for instance the implementation of the autonomous vehicle "Bertha": Ziegler et al, "Making Bertha Drive - An Autonomous Journey on a Historic Route", IEEE Intelligent Transportation Systems Magazine, 6 (2), pp. 8-20, 2014.

9 Proposal for a Regulation Of The European Parliament And Of The Council Laying Down Harmonised Rules On Artificial Intelligence (Artificial Intelligence Act) And Amending Certain Union Legislative Acts (COM/2021/206 final) (AI Act)

10 Memorandum to the AI Act, p. 8; Geminn, "Die Regulierung künstlicher Intelligenz", ZD 2021, 354.

possible technical approaches. The AI Act explicitly names both machine learning, logic, and knowledge-based approaches; statistical ones have also been mentioned as forms of artificial intelligence.[11]

9 These explicit regulatory considerations are at the front of recent technological developments. General German Private Law relies on statutes given in the German Civil Code. It had been enacted in 1900. It provides the fundamental rules of private law, which means particularly contracts and liability rules (e.g. torts). One may claim that—given the technological developments in the last 100+x years—the German Civil Code is technology neutral by design: it does not pose any explicit restriction on technologies to be used—particularly not on Artificial Intelligence.

10 However, the general structure of legal doctrines may affect different technical approaches in a different manner. Law and Economics scholarship has studied the effects that legal doctrine can have on society, in particular by providing a framework to enforce contracts and property rights effectively. Similarly, Law and Technology as well as Law and Innovation studies extended this approach to study the interaction between these fields.

11 Building on a Law and Technology approach, we study the effects of the liability regime on the choice between adopting a smart product on explicit rule representations and making use of machine learning methods.

12 We show that *correctness* as a desiderate of software engineering and the 'defect' in the legal sense are distinct. However, when safety-relevant features of a product are concerned, correctness of a software system is *de facto* the obliged outcome. If instead the manufacturer chooses to use Machine Learning technologies, thus merely approximating the desired outcome, the law may yield certain degree of inaccuracies. Finally, the question arises whether the law may dictate the use of explicit rule representations in cases where a certain output or behaviour is asserted or minimal guarantees hold.

11 In detail Geminn, "Die Regulierung künstlicher Intelligenz", ZD 2021,354. This commission states that these provisions are technology neutral: COM(2021) 206 final, 12: „as technology neutral and future proof as possible".

I. Two Tier-Perspective on Autonomous Agents

13 There are two perspectives on Artificial Intelligence as identified by Russell and Norvig: (1) the *behaviour* of the agent and (2) the *thought processes* or *reasonin*.[12]

1. Behaviour

14 The behaviour of an agent can be simply defined as the relationship between a certain input and the output. By 'output' it is meant any result of calculation that constitutes the agent's functionality. The 'behaviour' of an agent is usually what is of directly relvant to legal liability as the behaviour determines how the agent interacts with the environment and thus may be source of damage.

2. Reasoning

15 The *reasoning* corresponds with *how* a certain conclusion is being drawn.[13] It determines the steps the agent performs in order to ascertain the output. Any computer programme may be seen as a conditioned sequence of intermediate system states, and a concrete run of a system as an unconditioned sequence of system states. They can be invisible to the user.

16 By "intermediate states", I mean the sequence of states in between the output and input states. By evaluating the single steps taken by the agent, results might be traced and thus proven and explained.[14] This is invariant of the technology used. In classical algorithms, a sequence of system states is defined by the program flow. This is no different when machine learning comes into play. In neural networks, the *latent space* matches the single intermediate steps in the computation; in each layer there is some different representation of the input data which one may call a kind of interim result.[15]

12 Russell and Norvig, *Artificial Intelligence. A modern approach* (3rd Edition 2016), pp 1-2.

13 In logic, reasoning is being done by *inference:* propositions are being inferred according inference rules from a certain knowledge base: Russel and Norvig (fn 11), p 235.

14 For instance, the Hoare logic offers a formal-mathematical tool to prove an output (a postcondition) given a certain input (a precondition): Hoare, "An Axiomatic Basis for Computer Programming", 12 (10) Communications of the ACM, 576.

15 Cf. Lassance et al, "Representing Deep Neural Networks

17 These two conceptual tiers correspond with the terms of "specification" and "implementation". The specification of a system determines the outer behaviour given a certain input. The implementation determines the exact way a certain specification is being realized.

II. The Term Correctness of a Computer System

18 In Computer Science and Software Development, the term "correctness" refers to a behaviour of a computer programme. A computer system is correct if—given a certain input and certain preconditions in the state space—the *specified* preconditions hold, particularly the expected outcome.[16] The *specification* is a formal or informal description of what behaviour a computer programme is supposed to have.[17] Usually the term "specification" refers to both the requirements specification and the design specification. The first comprises the description of product behaviour in regard to the customer's *needs*. The latter is a more fine-granular description of the different components, modules, and interfaces (subsystems) of the system. Both are not representing the way *how* to achieve things, but *what* to achieve.

19 Functional requirements and non-functional requirements are still being distinguished on the specification side.[18] The functional requirements encompass that relation between input and output, respectively preconditions and postconditions. They describe the main functionality of the software. On the other hand, the non-functional requirements concern side-conditions, such as certain security standards, performance, etc.[19]

20 The implementation is the actual realization of the system, i.e., the concrete computer programme. The computer programme determines not only *what* behaviour a system may have (prescribed by the specification), but also it consists of concrete instructions to the system environment about *how* this behaviour shall be accomplished.[20]

21 Thus, on the one hand, from a Software Engineering internal perspective, the *correctness* is being assessed just by matching the implementation with the specification. From an *external* perspective on the other hand, a software product may be considered "sensible", "proper", etc. in regards to customer needs.

22 As described above, the specification describes the *behaviour* of an agent to its environment. The implementation is what constitutes the *reasoning* process, thus behaviour is reached by a specific sequence of instructions forming a certain sequence of states.

III. Implementation Approaches

23 Generally, there are two types of Artificial Intelligence approaches distinguished: Rule-based systems and Machine Learning methods.

1. Rule-based systems

24 Rule based systems belong to the group of "symbolic" AI methods. Symbolic AI relies on the use of logic and "ontologies" to represent knowledge.[21] The way behaviour is defined directly corresponds with the concepts of the problem domain. Thus, a rule "If A then B" can be directly represented using a certain *syntax*, e.g. "A → B", "IF A: B" etc. Ontologies can refine concepts as "A consists of 1 and 2", and semantic web methods may represent complex webs of relations between concepts.[22] For instance, one could represent legal rules symbolically by using

Latent Space Geometries with Graphs" <https://arxiv.org/abs/2011.07343>

16 Cf. Dennis, "The design and construction of software systems" in Bauer et al (eds.), *Software Engineering. An Advanced Course*, p. 22 "correctness of its description with respect to the objective of the software system as specified by the semantic description of the linguistic level it defines" The "description" in this sense is the code that describes the computer behaviour. The "objective" is what one can understand as the core of specification.

17 Schmidt, *Software Engineering. Architecture-driven Software Development* (2013), pp 93-111. *Bauer* et al, Software Engineering. An Advanced Course.

18 Cf Dick et al, *Requirements Engineering*, p. 172.

19 Critical discussion on this term in Glinz, *On Non-Functional Requirements, 15th IEEE International Requirements Engineering Conference* (RE 2007) DOI 10.1109/RE.2007.45.

20 Imagine a programme that shall sort numbers in descending order. In first year computer science classes students learn that there exist many different sorting algorithms (Bubblesort, Quicksort, Mergesort etc.). All of them are different implementations of the same.

21 These are called „knowledge-based agents" in AI research. Russell/Norvig, *Artificial Intelligence*, p 234.

22 For Semantic Web technologies used in the legal domain, see Benjamins et al, "Law and the Semantic Web, an Introduction", in: Benjamin et al (eds), *Law and the Semantic Web. Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications*, pp. 1 – 17.

a deontic logic, e.g. stating that somebody who murders another human being ought to be punished:

Murderer(x) → O(Punished(x))

25 If x is a murderer, he ought to be punished. It is clear to see that this representation of legal domain knowledge somewhat maps with the real life concepts behind it. In a rule-based system, therefore, behaviour of a computer system is being described *explicitly*. The language in which rules are being described *matches* the concepts of the problem domain; the domain-level concepts are being translated directly into logic-level names as predicates, functions, and constants.[23] The semantic *model* of the logic involved determines the truth of an individual sentence (rule) described.[24] The *model* thus maps the logical formalism (syntax) to the real-world concepts and the truth of sentences in the real domain.[25]

26 For *correctness* of such approach twofold conditions need to be satisfied. Firstly, the rule engine, i.e. the component that translates the rules into executable instructions, needs to be correct.[26] This encompasses both syntactic and semantic correctness; particularly the rules must be consistently interpretable.[27]

27 Secondly, the rule definitions themselves must be correct, thus leading to the correct behaviour of a system, given the rule translator works *correctly*. This means that rules shall conceptually map the problem domain the system is meant to represent.

28 However, there is non-determinism posing a problem because of the input/output operations of the autonomous system: the correctness property just implies that the programme meets certain post-conditions given a certain input meeting the pre-conditions. Neither it can be in any way logically proven that a person interacting with the agent meets the precondition of the system with their input, nor is it any possible to prove this for other input/output periphery as sensors. Reliability cannot be ensured in unreliable host environments.[28] Arbitrary changes in the circuits may inevitably happen and thus can lead to an error occurring.[29]

2. Machine Learning

29 Machine Learning relies on the idea that a certain model structure is parametrized and these parameters are being induced by a learning process.[30] The most common structure in modern machine learning is Artificial Neural Networks (ANNs). They are a layered architecture consisting of several computational layers, in which each layer is a linear combination of the previous layers, with some non-linear activation function applied on each output of the respective layer.[31] Whilst any neural network of the same architecture practically does similar steps, what constitutes the network solving a specific problem are the parameters (often referred to as 'weights'): in a simple ANN they are the real numbers that—simply spoken—determine the flow ratio of neurons of the previous layer to each of the neurons in the next layer.

30 This is a highly general and abstract way to solve a problem: the same general architecture can be trained to a theoretically infinitely high set of

23 For first-order logic rule representation Russell and Norvig, *Artificial Intelligence*, p 290.

24 Russell and Norvig, *Artificial Intelligence*, p 232.

25 The theoretical term *model* originates from logic to theorize the idea of semantic within formal systems. In Artificial Intelligence and Machine Learning, a *model* is something different: It is closer to the colloquial meaning of a *model* as an approximation of reality. However, they are related in the way that also a logical *model* is mapping reality semantics onto the finite syntax.

26 This maps what Dennis (fn. 15), p. 24 demands that for "host level descriptions [...] that are the result of automatically translating the designer's description, proving the correctness of the translator suffices."

27 See Morscher, *Normenlogik* (Paderborn 2012), p 117 ss for consistency in model theory.

28 *Dennis* (p. 24) calls this aspect 'reliability' in contrast to the correctness: A system is reliable if it may perform its functions in spite of any host system failure. A system cannot be entirely reliable if the host system may be fallible (p. 25).

29 It is suspected that cosmic rays may sometimes affect circuitboards and can randomly change the state of computer systems, see e.g. *Ziegler*, "Effect of Cosmic Rays on Computer Memories", [1979] 206 Science 776-788. It stands to reason that a certain degree of unreliability of computer systems is inevitable.

30 When talking about Machine Learning, a model is a combination of a certain shape of a network and their parameters. An architecture describes the principal ideas the model structure follows: For instance, sequences of input can be processed by Recurrent Neural Nets (*RNN*s), where the output of a model is 'plugged' back as a model input itself.

31 A linear combination is simply a somehow weighted combination (1,1,1) as can be calculated as linear combination with the weights (5,2,1) to (1*5+2*1+1*1)=5+2+1=8. Applied to n different weight vectors, one can create n different new values, which are output of the next layer.

problems, if enough training data is available. It can be proven that ANNs are universal approximators.[32]

31 However, the major shortcoming in practical use is, that it is difficult to explain what is exactly going on in the middle of this network, the so-called latent layers (as they are 'hidden' in the middle of the network). Nor can one prove properties of a neural network in general. This is often referred to as the "black box problem" of neural networks: whilst certain behaviour can be validated by testing, latent states (representing the reasoning process steps) are difficult to impossible to interpret.[33] The issue of "Explainable AI" is a current research issue, where these restrictions are aimed to be diminished.[34]

32 The most important property of these techniques is that they are merely approximative.[35] They will not be *correct* in the sense that they would always meet the right result given an input, if not all possible inputs have been tested. Testing every possible input will not be possible in most domains. Just imagine an autonomous vehicle that may be confronted with a sheer vast amount of possible traffic situations and their combinations.

3. Neuro-symbolic Integration

33 Several hybrid methods are aiming at combining both approaches to each other. They are known under the name "neural-symbolic integration". Essentially, networks may be used for for reasoning tasks and context understanding. Symbolic knowledge representations may be fed into a network, upholding certain properties of syntactic equivalence of the input logic.[36] However, if these architectures remain approximative approaches, they are neither provable nor totally correct.

B. Normative Knowledge vs. World Knowledge from a Legal Perspective

34 Before assessing the issue in more fine granular detail, we want to shortly discuss the importance of different types of knowledge that are to be represented in a system.

I. Knowledge Types

35 When talking about *knowledge* in context of AI systems, a rough distinction may be made between *world knowledge* and *normative knowledge*.[37] World knowledge is the set of propositions about the *being*, thus any states of or actions in the world. Normative knowledge is the knowledge about how the world *ought* to be; it can represent ethical or legal postulates.

36 From a mere information representation perspective, this distinction does not make a difference per se.[38] This is different in law itself. In criminal law, an important distinction between world knowledge and normative knowledge can be made. Whilst most criminal offences require an *intention* or *knowledge* of the factual circumstances that constitute the offence ("Vorsatz", mens rea), there is the principle "*ignorantia juris neminem excusat*".[39] According to the German Criminal Code, ignorance of the unlawfulness of an offence committed may only exculpate a defendant *not guilty* if the ignorance was *not avoidable*.[40] Regularly, there is everybody's obligation to obtain legal advice on acts whose legality is doubtful.

37 On the other hand, in private law (contracts and torts) an intention or knowledge of a wrongdoing is—according to legal scholarship as well as jurisdic-

32 Alpaydin, *Introduction to Machine Learning.* (4th edn, 2016), p 99.

33 Cf. Alpaydin (fn. 32), p 155.

34 *Gunning et al*, ORCID: 0000-0001-6482- 1973,. XAI-Explainable artificial intelligence. Science Robotics, 4(37). DOI: 10.1126/scirobotics.aay7120.

35 Cf. the 'probability risk' of artificial intelligence identified by Zech, "Liability for autonomous systems: Tackling specific risks of modern IT", in Lohsse et al., *Liability for Robotics and in the Internet of Things.*

36 E.g. Lamb et al., "Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective" <https://arxiv.org/abs/2003.00330>.

37 A finer distinction is made in Valente, "Use and Reuse of Legal Ontologies in Knowledge Engineering and Information Management" in Benjamins et al., *Law and the Semantic Web. Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications*, p. 71: They distinguish between different knowledge on the legal side. However, for the purpose at hand the more rough distinction will suffice.

38 However, Deontic (normative) Logic languages pose different issues on Computer Science than other logical systems. They do not touch the ways of representing, but of operating on them.

39 Ignorance of the law does not pose a defence; see Jackson, *Latin for Lawyers II*, (2014), p 166.

40 Section 117 German Criminal Code.

tion—considered to encompass both the knowledge of the circumstances that constitute the wrongdoing and its unlawfulness.[41] This difference to criminal law may be explained by the higher complexity of private law obligations; however it also stands out that in private law, most legal norms do not even require intention or knowledge of the unlawful act, but also let mere negligence suffice.[42] So the distinction is of less importance in private law.

38 For criminal law, the normative order imposes a dense obligation on everyone to inform themselves about the state of law. However, this becomes only relevant if one behaves against the law. Whilst the imagination of *factual circumstances* that fulfil the requirements of a criminal offense can cause liability for criminal attempt, the imagination of illegality of a behaviour that is not criminal, does not.[43]

39 Normative knowledge thus can have different legal implications than world knowledge. Put shortly, the law assumes that everyone must know about right and wrong, and failure to do so will not provide a defence against liability for malice.

II. Implications for Technical Systems

40 In current legal orders, there is no liability of technical systems themselves; any knowledge that is required for liability needs to be present in the human actors involved. For this constellation to occur, an analogy to § 166 German Civil Code is proposed:[44] If an autonomous agent took a decision "knowing" a certain fact (whatever this means for a computer system), then the human the agent connected to it cannot raise a defence of ignorance. This however is not widely accepted.[45]

41 Cf. Müko-BGB/*Grundmann* § 276 Rn. 158 ff.

42 § 826 German Civil Code is one of the rare examples where the law explicitly requires the intention or knowledge of the unlawful harm that triggers liability.

43 A maniac offense ("Wahndelikt") where the defendant just imagined that his behaviour was criminal does not form a criminal attempt and thus is not punishable. *Joecks/Kulhanek*, MükoBGB-StGB § 17 Rn. 38.

44 Recently Linke, „Die elektronische Person. Erforderlichkeit einer Rechtspersönlichkeit für autonome Systeme?, MMR 2021", 200 (with further references).

45 Against this, see only Cornelius, „Vertragsabschluss durch autonome elektronische Agenten", MMR 2002, 353 (355); Grapentin, *Vertragsschluss und vertragliches Verschulden beim Einsatz von Künstlicher Intelligenz und Softwareagenten*, 2018, S. 97.

41 For a machine there is no difference between "knowing" about the world and knowing about normative facts. It just behaves in the way it has been programmed. Thus, if active normative knowledge of a machine would matter, e.g. if there would exist a concept of malice done by a machine, there would not be any incentive of a programmer or operator to feed a machine with the normative knowledge (as then this would bar the responsible person from the defence of ignorance). The distinction between the knowledge of right and wrong and other kinds of knowledge should not be continued when considering autonomous agents from the legal perspective.

42 Generally speaking, the latent states of a machine (see above) are of no importance when considering the liability for a system. Only the behaviour matters. It does not matter *why* a machine takes a decision; both knowledge of fact and knowledge of norms only touch the question of personal responsibility of a human being. As long as computer systems themselves cannot be held accountable there is no need to distinguish between normative knowledge and world knowledge in autonomous agents *by law*. This does not mean that this distinction does not pose engineering problems when attempting to operate on formalized normative knowledge, i.e. by use of deontic logic.

C. Technical Correctness and Normative Standards

I. "Defect" in Product Liability

43 In the heart of the Product Liability Law regime lies the term "defect". Eliciting the scope of the term constitutes the remaining assessment of the problem.

1. Different "Flavours" of Defects

44 The EU Product Liability Directive establishes a liability for producers "caused by a defect in his product".[46] According to the definition given in the Directive, a product is defective, "when it does not provide the safety which a person is entitled to expect", taking into account the presentation of the product, the

46 Council Directive 85/374/EEC of 25 July 1985 on the approximation of the laws, regulations and administrative provisions of the Member States concerning liability for defective products (Short: Product Liability Directive), Art. 1.

expected use of the product, the time the product was put into circulation.[47]

45 It is acknowledged that this standard ought to be objective.[48] In the respective recital of the German implementation of the Directive, it is explicitly stated that it relies on the "expectations of the public"[49], which is to be concretised as the usual circle of ideal users.[50] This means it relies on the expectation of the product's target group. However, some call the wording "expected safety standard" an empty formula, as it did not make it any easier for courts to ascertain the standard of safety.[51]

46 Jurisprudence has delivered more concrete formulas. For instance, the level of the product's safety standard to be expected is ascertained by an "exhaustive consideration", taking into account the size and scope of the dangers, the cost of safety measures as well as further circumstances as the detectability and avoidability of dangers.[52] Generally, the manufacturer was only liable for security measures whose cost was reasonably proportionate to their utility.[53] This "risk-utility-test" is also the formula to determine the safety standard under U.S. law.[54]

47 For the separate types of defects, doctrine distinguishes between those of design, manufacture, and instruction. When considering software systems, on which it is at least partially acknowledged that product liability law is applicable,[55] it also considers how the safety standards connect with the term "correctness".

47 Product Liability Directive, Art. 6.

48 BeckOGK/*Goehl*, § 3 ProdHaftG Rn. 14.

49 BT-Drs. 11/2447, 18.

50 BeckOGK/Goehl, § 3 ProdHaftG Rn. 15.

51 MükoBGB/Wagner, § 3 ProdHaftG Rn. 7.

52 BeckOK-IT-Recht/*Borges*, § 3 ProdHaftG, Rn.8.

53 MükoBGB/Wagner, § 3 ProdHaftG Rn. 7; BGHZ 181, 253 Rn. 23.

54 Geistfeld, "A Roadmap for Autonomous Vehicles. A Roadmap for Autonomous Vehicles: State Tort Liability, Automobile Insurance, and Federal Safety Regulation" (2017) 105 California Law Review 1611.

55 At least for embedded systems (software that has been integrated into a physical good) this is acknowledged: MükoBGB/Wagner, § 2 ProdHaftG Rn. 6. However, this should not be discussed another time in this paper.

48 First, it is obvious that these terms are of different meaning. By definition, a software is correct if it matches the specification.[56] Now, given the specification also matches with the safety standards demanded by law (including the safety standard demanded by a reasonable and ideal user), a *correct* software also fulfils the safety standards demanded by law. In this case, one can state the presumption that *correctness* is a prima facie condition for a software to fulfil these safety requirements.

49 However, neither an incorrectness implies a defect necessarily, nor follows from a defect in the legal sense that the software is technically incorrect. Literature restricts the term "defect" to features that are "safety relevant".[57] This can be explained by the purpose of Product Liability Law: there shall not be an obligation to deliver an optimal product.[58] Product Liability is about safety only. Therefore, naturally not every incorrectness poses a defect.

50 On the other hand, a software may be completely correct, but still not meeting the product safety requirements. The flaw is therefore to be found in the specification. It might be that the requirements are itself "incorrect" or "flawed". This only applies to the "external" safety expectations that cannot be systematically captured within the "internal" development sphere that is only concerned with matching the implementation with the specification. Whereas, the flaw can be that *needs* have not been sufficiently put into *specification*, which means that the product does not fit the customer *needs*.[59] From an engineering perspective, it is to be said that all customer needs are required to be taken into account when eliciting requirements; they come in vague statements from the persons in charge of eliciting the needs.[60] This will entail observing the market and also the legal framework around this market, particularly safety standards.

2. Is always correct software expected?

51 Imagine a judge examining a case of a potentially flawed feature that is safety-relevant. Without doubt, this leads to an application of the product li-

56 See above, p 5.

57 MükoBGB/*Wagner*, § 3 ProdHaftG Rn. 2.

58 BeckOK-IT-Recht/*Borges*, § 3 ProdHaftG Rn. 21.

59 In any requirements elicitation process the (abstract) *needs* serve as "input requirements" to the next level of requirements elicitation. Dick et al (fn. 17), p. 33 ss.

60 Dick et al (fn. 17), p. 33 ss.

ability regime. The question then is whether every incorrect implementation of a safety-relevant feature triggers liability. By the term *incorrect* I mean that the specification of the feature is flawless; the engineers in such a case correctly considered a feature that falls into the scope of the public safety expectation. The defect to be considered merely lies in the wrongful implementation.

52 It is highly doubtable whether the public expectation always demands software to be *correct* in the terms stated above.[61] Obviously, this cannot be determined generally and depends highly on the requirements of the domain. From an algorithmic perspective, there are some problems that are so-called NP-hard: a *correct* solution needs—from what theoretical computer science's complexity theory is at least presuming—exponential runtime complexity.[62] Thus, they cannot be practically solved correctly as the runtime would be too high.[63] An example is the Traveling Salesman Problem (TSP), where the shortest path in a graph is searched, that traverses all nodes and finishes at the starting point.[64] It cannot be solved efficiently (which means in polynomial / non-exponential time) whilst being correct. However there exist heuristics, that do not guarantee an optimal solution, but a reasonable runtime.[65]

53 Therefore, the public safety expectation (and this is only what matters)[66] cannot be an always correct software, even in safety-relevant matters; if complex problems are solved that can only be solved by approximating algorithms, there cannot be claimed a reasonable expectation of a correct software. Then, however, testing needs to be done to a reasonable extent.

54 However, a manufacturer cannot always claim the impossibility of a correct implementation. There are cases where a product cannot be safely brought to market, and thus shall not be issued at all.[67]

55 In parallel to this test, side-constraints posed by legal rules and standards must also be taken into account.[68] For autonomous vehicles, the German Traffic Code (*Straßenverkehrsgesetz*) imposes a regime for the technical admission requirements. Thus, the law specifies that any autonomous vehicle ought to ensure the behaviour of a "risk-minimal" state: A vehicle ought to set itself to a safe idle mode in a safe position (§ 1 d para 4 StVG, § 1 e para 2 no 3), or otherwise an infringement of traffic rules would occur. This is an explicit *minimal guarantee* of the product safety standard by law.[69] It is to be further discussed whether these *minimal guarantees* demand a correct implementation or can be implemented by approximation methods.[70]

3. Software Defects as Defects of Design only?

56 From an engineering perspective, a system may be either incorrect (i.e. its implementation does not meet the specification) or suffer of poor specification and thus the requirements are badly elucidated and do not meet the customer needs. Generally, one could speak of a *defective product* in this sense.

57 An issue however is to decide whether a defect is legally a design or manufacturing defect. This distinction is necessary as it determines the well-known *safety standard test*: defects of design are determined by actually applying the *risk-utility test* while defects of manufacture on the other hand can be proven by showing that the individual exemplar suffers of a disadvantageous deviation from the design plans.[71] This is because the public may rely on the specific properties of a product series.[72] The blueprints of a product thus pose a self-inflicted

61 Cf. BeckOK-IT-Recht/*Borges*, § 3 ProdHaftG Rn. 21; Taeger, „Produkt- und Produzentenhaftung bei Schäden durch fehlerhafte Computerprogramme" 1995 Computer und Recht 257, who stress that flawed software does not pose a defect necessarily.

62 The "P=NP-Problem" is actually a Millennium Problem for which the Turing Society offers a prize of One Million Dollars. Solving this problem would go beyond the scope of this essay. It may be solved in a further paper by the author. See Goldreich, *P, NP, and NP-Completeness. The Basics of Computational Complexity*, p. 48 ff.

63 Goldreich (fn 61), p. 50.

64 Lin and Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem" [1973] 21 (2) Operations Research p 498-516.

65 Lin and Kernighan (fn 61).

66 BeckOK-IT-Recht, § 3 Rn. 21.

67 BGH NJW 2009, 2952; BeckOGK/Goehl, § 3 ProdHaftG Rn. 15; MükoBGB/Wagner, § 3 ProdHaftG Rn. 45.

68 MükoBGB/Wagner, § 3 Rn. 27 ff.

69 The term "minimal guarantee" refers to software specification, in which the expected behaviour of a system or subsystem is stated, disregarding of a successful or non-successful execution of the component. See fn 101.

70 See below, p 19.

71 Wagner, AcP 217 2017, 707 (725 s).

72 BeckOGK/*Goehl* § 3 ProdHaftG Rn. 70.

safety standard that may be stricter than the objective standard matching the public expectation applying in the case of a design defect.

58 A manufacturing defect is a disadvantageous deviation of the product from the safety standard imposed by the producer themself.[73] In literature, Wagner claims that manufacturing defects of software only comprise wrongful *delivery* of software to individual specimens of the product, mainly relating to embedded systems.[74] One can reasonably doubt whether this perspective is entirely correct. Wagner further claims that a software not meeting the respective safety requirements was "*per definitionem*" suffering of a production defect, as every specimen of the product was affected.[75] However, public expectations may also arise from certain specifications that represent standards shared by several producers of software (*interfaces*). This comes into play particularly when components are delivered for end-user software products. Therefore, unlike Wagner's claims, incorrect software may pose a production defect rather than a design defect if one considers the coding as part of fabricating an end product rather than just constructing it.

59 In the analog world, a defect of design may be considered as wrong *blueprints*. They can be regarded as what specifications are for the manufacture of software. If a software is *incorrect* as it was not matching the specification, it is comparable to an item that has not been produced according to the blueprints. It is—from this perspective—a defect of manufacture. On the other hand, a wrongful specification resembles a defective blueprint. It stands to reason that—if the manufacturing defect's *differentia specifica* is the deviation from the *intended design*[76]—incorrect software deviating from the specification would have to be regarded as suffering from a manufacturing defect.

60 This is particularly important when software components are being delivered. The specification fulfils a special task in multi-component software systems. It defines the *interfaces* with which other components may communicate with the respective sub-system or component.[77] A component of a software may be a product itself in the sense of Product Liability Law.[78] Now if a component promises *by specification* to deliver service to another host environment this specification serves as much as a self-inflicted standard as a blueprint in a series of fabricated goods does. Public expectations are then subjectively formed by the intended design.

61 I do not want to argue out this issue; there may be good arguments for not considering incorrectness of software as defect of manufacture, certainly. It is not just as simple as to refer to the argument of a *per definitionem* nature of the implementation process. It highly depends on the mapping of analogies from the digital to the analogue. In literature it has therefore been proposed—with similar arguments—a new type of defect, the "generic manufacturing defect".[79]

62 Finally, it cannot be predicted today that the prevailing opinion on the nature of a bug will be seen correctly as a manufacture defect, if the defect relies on a deviation from publicly available interface specification. I will thus assume for the purpose of this study that incorrectness will lead to a defect of design rather than manufacture.

4. Proving versus Testing

63 To ascertain the quality of a software product, the two main ways are *proof* and *testing*. A proof is a mathematical (or other formal) procedure in which the logical necessity is induced, that a software or an algorithm returns the correct output (or sets the machine into the specified state) given a certain input.[80] For this it is necessary to observe the software's code. Formal proving is considered more of

73 Cf. MükoBGB/*Goehl* § 3 ProdHaftG Rn. 70; discussed by *Hubbard*, Sophisticated Robots: Balancing Liability, Regulation, and Innovation, [2015] 66 Fla. L. Rev. 1803 (1854 ss).

74 Wagner, *Produkthaftung für autonome Systeme*, AcP 217 (2017), 707 (725 s).

75 Wagner (fn. 74), AcP 217 (2017), 707 (725 s).

76 Turner and Richardson, "Software defect classes and no-fault liability." UC Irvine. ICS Technical Reports. Published 1999-04-05 p 16 <https://escholarship.org/uc/item/11v8f8tc>.

77 Foster and Towle, *Software Engineering. A Methodical Appoach* (2nd Edition 2022), p 194.

78 § 2 Produkthaftungsgesetz regards as product also the items that are part of another product. This relies on Art. 2 Product Liability Directive. Similarly Art 3 Product Liability Directive considers the manufacturer of a component as producer.

79 Turner and Richardson (fn. 78), p 19 <https://escholarship.org/uc/item/11v8f8tc>.

80 Dennis (fn. 76), pp 22 ff: "To prove correctness of a software system or component, one establishes by logical deduction that some description of the system or component asserted to be correct by the designer is equivalent to the description of the system or component expressed at the host level". The "description of the system or component asserted to be correct" is none less than the *specification*.

a theoretical thing.[81] Particularly, every computer programme entails a sort of non-determinism, as a software usually works in an operating system environment with a very large state space; the programme calls input/output functions indirectly by system calls to the operating system, and usually user inputs are not foreseeable. In short: one cannot make sure that the executing environment satisfies all the preconditions specified.[82]

64 Furthermore, even in a very simple programming language, it can be shown that the so-called Turing-completeness leads to the *undecidability* of certain properties of the code.[83] The well-known Halting Problem states that for no programming language that enables loops or recursions (possibly leading to infinite loops or recursions), there can be a program that decides *for all valid programs* whether this program falls into an infinite loop or recursion. Thus, there will never be any algorithm, software, or Artificial Intelligence that can cross this logical barrier. However, this does not mean that programmes cannot be written in a form that enables a proof on their correctness. This process just cannot be automatized.

65 Machine learning applications cannot be proven so far; we would have to understand what is going on inside of the model. Instead, only statistical margins can be defined, that a machine learning system shows a certain behaviour (given a certain input) with some percentage of probability.[84] This is done by means of testing. The term binary term *correctness* may then be replaced with scalar measure of *performance* of a model. Therefore, a programme is either correct, or it is not, *tertium non datur*, but it can be performing well (by accuracy metrics, e.g.) more or less.

5. Impacts on Product Liability

a) Correct Boundaries of Decisions and Training Procedures

66 Originating from American law, the consumer expectations are being ascertained by a "risk-utility test".[85] A product is thus to be considered defective if it poses risks to the consumer that are not being outweighed by the benefits.[86] Marchant and Lindor argue that this leads to a prohibitive effect of further developments as every advantageous improvement of the algorithms used can thus create liability, as the benefits of implementing such a change (particularly protecting human life, in the example of autonomous driving) would outweigh the cost, at least when highly valuables as life and body are endangered.[87] This would lead to basically any bug imposing liability.

67 Geistfeld correctly objects that this argumentation relies on the assumption that autonomous cars are being explicitly coded by rule definition.[88] Instead, he distinguishes parts that concern "rules that *constrain* or *guide* the machine learning, such as coding that instructs the vehicle to always stop at stop signs"[89] and the parts that make use of machine learning technologies.[90] Only the former was subject to a code-evaluation as proposed by Marchant and Lindor.

68 First of all, it needs to be stated that—given Marchant and Lindor are right with their claim—*correctness* in the sense stated above would be a minimal requirement for autonomous driving in regard to executive driving functions that—from the German perspective—represent safety-relevant features of an autonomous car (given the behaviour demanded by law was flawlessly specified). Thus, to avoid liability a manufacturer has to carefully (mathematically) *prove* both the rules' correctness and correctness of the piece of software that interprets the rules.

81 For instance, first year CS students are being taught the Hoare Logic (fn. 13) to prove that certain conditions hold given a certain preconditions by analysing the source code of a programme.

82 This is being called a problem of "reliability" of a software system: *Dennis* (fn. 15), pp 24 ss.

83 Enderton, *Computability Theory* (2011), pp 79-102.

84 Leupold et al., *Münchener Anwaltshandbuch IT-Recht* (4th edn 2021), 9.1 Rn. 12.

85 Geistfeld (fn. 85), pp 1642 s. In German law the *Bundesgerichtshof* has accepted this notion for their own adjudication.

86 Geistfeld (fn. 85), pp. 1642 s.

87 Marchant and Lindor, "The Coming Collision Between Autonomous Vehicles and the Liability System"(2012) 52 (4) Santa Clara Law Review 1321, pp 1334

88 Geistfeld (fn. 85), p. 1644.

89 Geistfeld (fn. 85), p. 1645.

90 Geistfeld (fn. 85), p. 35.

69 If this is being restricted to the explicit "rules that constrain or guide" the machine learning (as Geistfeld claims), it remains that both correctness of the machine learning routines themselves (training algorithms) as well as subroutines enforcing certain behaviour as layer on top of the learned behaviour ought to be *correct* for evaluating the product as defect-free.

70 Geistfeld does not go into the existence of methods that are in-between both approaches. They already have been introduced as "neuro-symbolic integration".[91] Roughly, rule representations are being used to influence the training to converge into a certain direction.[92] The system itself remains however approximative.[93] Therefore neuro-symbolic integration is not *correct* in the sense defined above. If a manufacturer makes use of these approaches, it is to claim that *at least* the rules injected into the machine learning model need to be *correct*, thus being a valid representation of the specified behaviour. This notion of *correctness* entails a very isolated, narrow view on the "linguistic level"[94] of the rule definition language, and not the behaviour of the entire system. In this case also, sufficient pre-market testing is the only means to decrease the risk of liability when using still-approximative "neuro-symbolic integration".

b) Escape to Approximations

71 Basically, developers of autonomous cars are free to decide which technical approach is to be used. However, when making use of machine learning technology, this means that a manufacturer would in fact *opt out* the explicit code evaluation done with the liability test. Instead, they would opt for merely ensuring sufficient pre-market testing rather than a mathematical proof of correctness. However, this may lower standards, as correctness of a software will not be necessary. There could be a race to the bottom of quality standards by an escape of developers to mere approximations.

72 Thus, it is problematic that there can be an arbitrary choice between the approaches. Approximative solutions may only be acceptable if the risk-utility test allows a system to be merely approximative—in the case that a correct solution would be either too expensive to obtain or computationally intractable. If the manufacturer opts for approximative solutions, it is to make sure that the system had been sufficiently tested, with regard to the risks it poses.[95]

73 If the manufacturer uses the explicit rule representation approach, the question is whether any coding error (bug) would pose a defect that the manufacturer is liable for. This is being argued by Marchant and Lindor who claim that given the risk-utility test, in risky domains *any* bug would impose less cost to remove than the risks to be expected if the bug would remain in the system.[96] This again would carry a legal obligation for the manufacturer to ensure *correctness* of the explicit rule implementation, regarding safety-relevant features. If certain behaviour is steadily specified, mere approximations to achieve this behaviour will not suffice.

74 Moreover, the largest burden of debugging lies in the *identification* of bugs. However that identification costs are part of the trade-off between risk and utility in the respective test to ascertain a defect is doubtable: In the Directive[97] there is a distinction made between the *identifiability* of a defect and the *implementability* of safety standards. Whilst the question of implementation cost touches the question of an expected safety standard,[98] the non-recognisability of a given defect is merely a defense as provided by § 1 Abs. 2 Nr. 5 ProdHaftG.[99] The prerequisites of the defense of non-recognizability of a defect are much stricter and do not admit a risk-utility-test. It stands to reason that courts will never consider a bug as not identifiable. According to the "state of science and technology" a bug could always be considered identifiable. And if a bug has been identified, the effort it costs to solve it is marginal most of the time. The risk always outweighs the burden.

75 This leads to the proposition that, when using rule-based approaches, it is possible that—due to the strictness of the risk-utility test —making use of explicit rule definitions may lead to higher liability risk. The disproportionate cost to review code for bugs may not help the manufacturer to argue a case

91 See above, p 8.

92 See above, p 8.

93 See above, p 8.

94 This is how Dennis defines a logical level of a software, on which correctness applies: *Dennis* (fn. 15), p 14.

95 This is stressed by Geistfeld (fn. 85), p 1646.

96 Marchant and Lindor (fn. 87), p 1334.

97 See Council Directive 85/374/EEC of 25 July 1985 on the approximation of the laws, regulations and administrative provisions of the Member States concerning liability for defective products, Art 7.

98 And thus is a question of § 3 ProdHaftG resp. Art. 7 lit e of the Directive.

99 Cf. MükoBGB/Wagner, § 1 ProdHaftG, Rn. 52

for themselves in the course of the risk-utility test. Therefore, when using explicit rule-based methods to implement a software, the law will *de facto* require *correctness* of this system, if they potentially affect safety-relevant features. In particularly safety-critical domains, most features are safety-relevant indeed.

76 On the other hand, whether the cost of *testing*, when using approximative machine learning approaches, belongs to the cost of identification of a defect and not the cost of implementation is doubtful. In any case, the obliged scale of testing would depend on the "state of science of technology" in the way that the testing procedures need to be in accordance with the state of the art of computer science, and the scale of testing sufficient to ensure a reasonable safety standard. This also depends on available computational power.[100] Testing therefore will always remain imperfect, and no "perfectly" tested system can be demanded by law (which would mostly not be even possible). The latter case means a necessary trade-off between the cost and benefit of safety measures; this is a strong argument to position the question of scale of testing (particularly how many test runs and how much test data is needed) to the less strict question of expected safety standard.

77 It seems therefore that by using machine learning techniques, the manufacturers can avoid their liability for correctness of a system; the law may tolerate system failures for machine learning systems more than if explicit rules have been used. This appears to be an adverse effect as it might lead manufacturers to escape strict code evaluation by opting for approximative approaches!

c) Minimal guarantees and safeguards

78 An exception to the principle of free technical choice may arise if the law demands that certain behaviour should occur in any case, thus with a probability of 100 percent. For instance, Leupold and Wiesner assert that the absence of "decision boundaries" may lead to product liability.[101] Geistfeld similarly recognizes that in autonomous driving environments, there would—at least—exist explicit "rules that *constrain* or *guide* the machine learning, such as coding that instructs the vehicle to always stop at stop signs"[102]

79 With "decision boundaries" it is meant a fixed range in which a system can autonomously decide but may never go beyond these boundaries. An autonomous car may be coded in the way that e.g. the *Acceleration module* may not exceed a certain velocity. By our nomenclature, this is rule-based coding rather than machine learning as the behaviour will be explicitly defined, and the cap of velocity not just be induced by prior training data. Such boundaries may be imposed by law or by technical standards, or just arise from technical necessity. As rule representations, these boundaries ought to be correct as well if they concern safety relevant features.

80 Aside from that, there may be *minimal guarantees* to be expected. This is behaviour that should in any case hold and should be guaranteed by a system even in case of operation failure.[103] The German regulations give an example of the admission of autonomous vehicles. The law explicitly demands that a system should

> [...] set itself into a risk minimal state, if the driving may only be continued with an infringement of traffic rules.[104]

81 This kind of provision will also oblige the manufacturer to implement such a safeguard functionality; legal safety requirements can be expected to be satisfied by the public. Now the question would arise whether the manufacturer could merely implement this behaviour by training the system to behave this way (which would mean as last resort before an infringement of traffic rules, drive to the right and stop!). Against this it can be argued that the law requires such behaviour to be implemented correctly, so that a mere approximation by machine learning techniques would not suffice.

82 One may argue that the existence of a minimal guarantee does lead to a legal obligation to ensure that the asserted behaviour shall be triggered in any case possible, thus with a probability of 100 percent given certain prerequisites. This could only be achieved by explicit rule representation,[105] as this

100 Moore's law states the monotonic, exponential growth of transistor size and thus computational power (cf. Kurzweil, *The law of accelerating returns*, <https://www.kurzweilai.net/the-law-of-accelerating-returns>). Thus, the technical developments will also shift the standards for the adequate scale of testing to more intense testing.

101 Leupold/Wiesner, 9.6.4, Rn. 26.

102 Geistfeld (fn. 85), p. 1644.

103 This is a term used by to set such behaviour of a computer system within a Use Case; thus it originates from the requirements elicitation phase: Cockburn, *Writing Effective Use Cases*, p. 83.

104 § 1e II Nr. 3 StVG.

105 Of course, this 100 percent would be anyway conditioned on full reliability of the host system.

behaviour merely being induced by training data would never be an optimal solution. However, whilst it is possible to ensure correctness, reliability affects the product behaviour as well. Reliability means a stable system behaviour despite any hardware or subsystem error. It stands to reason that an autonomous driving system will always be prone to hardware errors and thus the perfectly reliable system does not exist.

83 One may say: At least, if there is no 100 percent safety, one should at least expect optimal safety. This would mean that a *correct* implementation of the feature can be expected, and this would bar the manufacturer from using approximative methods for the feature.

84 Against this, it may be argued that such strict standards do not apply to other, non-digital products. For a conventional car, one would assert that its brakes should be effective. Obviously, there is always a probability that the brake fails, there cannot be 100 percent safety. Unlike computer code that works in a *conceptually* perfect environment (correctness assumes that the computer does what it is being told to do), mechanical parts are not considered to work in such a formal machine environment. Why would a prerequisite of *correctness* be made for certain features in a digital system, but not in other, analogous system? The doctrine of risk-utility test gives the answer to this question: because *it is usually feasible* at proportionate cost. If the minimal guarantee cannot be implemented effectively, the system would be usually too risky to be published, or an approximative solution would suffice.

85 This depends on the individual case matter. As a rule of thumb one can state:

> Features that are mandated by law to exist shall be explicitly coded (by a rule).

86 Therefore, a manufacturer may not lawfully refrain from explicitly representing *guaranteed* behaviour; an arbitrary escape to approximative solutions is not possible here. However, it is an individual question of legal statute interpretation of the safety standards demanded by law whether it imposes an actual minimal guarantee on the manufacturer, or just aims at ensuring a very careful consideration of a certain safety aspect.

D. Regulatory Impact of the AI Act

87 Interestingly, one cannot find the term "correctness" in the "AI Act" proposal. Instead the term "accuracy" is used for postulating in Article 15, para 1 that systems ought to achieve an "appropriate level of accuracy" (cf. Rec. 38, 47, 49). This wording appears to imply that the regulator acknowledges the fact that machine learning will only be accurate to a certain degree, thus is restrained to approximations. What is an appropriate degree of approximation, remains unclear and will depend on the single case as intended.

88 However, the transparency requirements of Article 13 para 1 of the proposed AI Act may impose a stricter constraint on the design choice:

> "High-risk AI systems shall be designed and developed in such a way to ensure that their operation is sufficiently transparent to enable users to interpret the system's output and use it appropriately. An appropriate type and degree of transparency shall be ensured, with a view to achieving compliance with the relevant obligations of the user and of the provider set out in Chapter 3 of this Title."

89 "Sufficiently transparent" sounds rigorous given that interpretability of the state-of-the-art machine learning technologies is still in its infancy. For certain high-risk systems this might mean that only explicit rules may be used so that the system can output a reasonable explanation.

90 The manifest itself is even more generous in its understanding of transparency:

> "Users should be able to interpret the system output and use it appropriately. High-risk AI systems should therefore be accompanied by relevant documentation and instructions of use and include concise and clear information, including in relation to possible risks to fundamental rights and discrimination, where appropriate"[106]

91 It does not state that the user shall be allowed to interpret from the latent space of the machine learning model what *explainability* is about from the technical perspective. The manifest appears to let documenting the caveats of a system suffice, particularly its approximative nature. This is the key information that is needed in order to interpret an approximative system's output and to estimate its significance. Whether this is enough to achieve their intended goal remains questionable. Society might still rely on non-transparent models while being aware of the mere correlation-based stochastical nature. The general idea of a mere disclosure or information-based approach rather than substantive regulation would generally be welcome. But then the Commission could not evade the question of why it opted for the rather substantive regulatory approach for the rest.

106 Manifest of the AI Act rec. 47.

E. Contract Law – The Digital Content Directive

92 Similar conclusions as for the product liability may be made for the field of contract law. According to the Directive, digital content providers are obliged to fit the contractual requirements (subjective requirements) as to functionality, compatibility, interoperability, and other features.[107] It demands that beneath these subjective requirements the product should be fit for the purposes for which digital content or digital services of the same type would normally be used, taking into account, where applicable, any existing Union and national law, technical standards or, in the absence of such technical standards, applicable sector-specific industry codes of conduct.[108]

93 It is important to stress that these requirements are not restricted to safety requirements as the product liability regime is. It goes beyond them and also comprises any reasonable expectation of the customer to get a fully functional product. To be free of system failures is also a question of whether the system is secure.[109] In Information Technology, a secure system needs to be reliant and available; unreliance and unavailability may originate from both software bugs as well as human manipulation.

94 Any code incorrectness that leads to system failure in this sense may form a breach of contract. However, there is no equivalent to defective production as the measure is either by contract or inflicted by the industry average. It is yet to ascertain whether a risk-utility test will be applied.

F. Conclusions

95 Manufacturers should be aware that if they use rules to represent knowledge and behaviour, they ought to be correct! By making use of machine learning techniques manufacturers may partially avoid code assessment in the course of a dispute and thus may diverge from a strict correctness prerequisite. Then they simply need to provide evidence for sufficient testing before the product had been put on the market. However, a caveat is formed by minimal guarantees to be implemented—they ought to be implemented explicitly. If they are not computationally tractable or just way too costly to implement this can bar the manufacturer from putting a product on the market. It seems that under current liability law not all smart agents are created equal; approximative solutions are not required to be assessed as harshly as when explicit algorithms are used.

G. Acknowledgment

96 The research leading to these results is funded by the German Federal Ministry for Economic Affairs and Energy within the project "KI Wissen – Automotive AI powered by Knowledge". The author would like to thank the consortium for the successful co-operation.

107 Article 7 lit a. Digital Content Directive.

108 Article 8 para 1 lit a Digital Content Directive.

109 Compare Recital 42 of the Digital Content Directive.