machine learning & knowledge extraction

MDPI

Article

Challenges of Machine Learning Applied to Safety-Critical Cyber-Physical Systems

Ana Pereira * and Carsten Thomas

School of Engineering, Hochschule für Technik und Wirtschaft Berlin, Wilhelminenhofstraße 75A, 12459 Berlin, Germany; Carsten.Thomas@htw-berlin.de

* Correspondence: Ana.Pereira@htw-berlin.de

Received: 23 October 2020; Accepted: 17 November 2020; Published: 19 November 2020

Abstract: Machine Learning (ML) is increasingly applied for the control of safety-critical Cyber-Physical Systems (CPS) in application areas that cannot easily be mastered with traditional control approaches, such as autonomous driving. As a consequence, the safety of machine learning became a focus area for research in recent years. Despite very considerable advances in selected areas related to machine learning safety, shortcomings were identified on holistic approaches that take an end-to-end view on the risks associated to the engineering of ML-based control systems and their certification. Applying a classic technique of safety engineering, our paper provides a comprehensive and methodological analysis of the safety hazards that could be introduced along the ML lifecycle, and could compromise the safe operation of ML-based CPS. Identified hazards are illustrated and explained using a real-world application scenario—an autonomous shop-floor transportation vehicle. The comprehensive analysis presented in this paper is intended as a basis for future holistic approaches for safety engineering of ML-based CPS in safety-critical applications, and aims to support the focus on research onto safety hazards that are not yet adequately addressed.

Keywords: machine learning; safety; cyber-physical systems; hazard analysis

1. Introduction

Cyber-physical systems are systems that continuously interact with the physical world and human operators. Combining the cyber and physical worlds allows the emergence of technologies, which fosters innovation for a broad range of industries. The Cyber-Physical Systems (CPS) community has been experiencing a strong push towards integrating machine learning in their systems. Besides the large amount of analyzable data [1], machine learning algorithms help to master control problems that do not lend themselves to traditional algorithmic control approaches. CPS applications include medical devices and systems, assisted living, traffic control and safety, advanced automotive systems, avionics and aviation software, critical distributed robotics, manufacturing, among others [2], where most of these applications are safety-critical.

Industrial CPS-based applications have also been developed and deployed in Industry 4.0 [3,4]. Through CPS, Industry 4.0 (also referred to as Smart Factory [5]) is focused on creating smart products, procedures and processes [6]. In the Industry 4.0 context, a proper safe collaboration between humans and machines is needed, due to the widespread usage of Machine Learning (ML)-based robots (industrial-, service-, mobile-robots, such as Autonomous Intelligent Vehicles (AIV)) in factories [7–9]. These CPS incorporate ML components in order to perform complex tasks such as pattern and image recognition (i.e., tasks difficult to effectively perform using algorithmic methods alone) [10]. Along with its growing applications, safety aspects of machine learning became a crucial topic of research [10,11]. For most real-world applications, the safety of ML-based systems must be evaluated and assured.

The problem of failures in machine learning systems, defined as unintended and harmful behavior, may emerge from inaccurate incorporation of ML into systems [12]. The probabilistic nature of ML algorithms sometimes conflicts with the safety culture adopted when developing a safety-critical system [13]. Their engineering is less well understood than of general software: despite a relatively static and clean structure, their functionality depends on numerical parameters that are extracted from datasets [14], without being explicitly programmed to perform a specific task [15].

The growing interest in the area of safety of ML has brought together researchers from the two areas, in order to bridge the gap between safety engineers and machine learning [11]. Concerns about assurance and certification of such systems have been explored in the literature taking different approaches (high-level vs. low-level) [16,17]. Despite the advances, there is still an ongoing discussion in the community on the definition of which standards and norms should be defined for the certification of ML-based systems. Several recently launched initiatives on certification, such as the Dependable and Explainable Learning (DEEL) Certification Workgroup [18] in France and standardization working groups, for example, EUROCAE WG-114 and SAE G-34, indicate the relevance of this topic [19].

Our paper presents an extensive detailed analysis of the characteristics of ML models that could potentially affect safety. This is achieved by describing the activities that occur during the machine learning lifecycle (i.e., the iterative process responsible for designing and implementing a machine learning component) that could introduce a hazardous behavior. We illustrate the process using a real-world application scenario, i.e., a self-driving vehicle that operates in factory warehouses sharing transportation routes and tasks with human workers. We consider that this illustration is extremely important to support a clear view of the content of the paper. The aim of our study is two-fold: (1) supporting the identification of sources for potential hazardous behavior in the machine learning lifecycle, detailing how it can affect the overall system and illustrating it with an example scenario; (2) establishing the basis for a future definition of a coherent framework to integrate hazards analysis and verification techniques, along with mitigation mechanisms (detection and removal), towards the ultimate goal of the standardization of the development process of safety-critical ML-based systems.

The rest of the paper is structured as follows. In Section 2, we describe the main literary works that we consider fundamental for the advances made on the safety of ML so far, ranging from the definition of safety concepts to the steps already taken into certification and assurance of ML. In Section 3, we present some background on the concepts of safety as well as machine learning, and ultimately, on safety and machine learning together. Section 4 extensively details the ML lifecycle following a structured approach, defining activities and hazardous behavior of each phase of the cycle. Section 5 finalizes the paper by highlighting our main contributions and potential future steps.

2. Related Work

Recently, there is a growing concern in defining safety concepts, principals, and standards for ML-based systems, especially for safety-critical applications. An existing community of researchers have been studying the safety of ML-based systems that interact with the physical world [12].

Due to the variety of applications and also to its relatively recent research, authors have focused on different topics regarding the safety of machine learning. The first steps of Amodei et al. [12] detail concrete problems of ML models in order to avoid accidents, especially for the case of reinforcement learning. On the same topic, Faria [11] addresses safety in machine learning, focusing not only on reinforcement learning, but also on supervised and on non-supervised methods. Both researches describe the identified failure modes, and both present strategies for managing safety issues. Furthermore, similar works were presented by Varshney et al. [20,21], where both papers focus not only on defining machine learning safety concepts, but also on the safety strategies for specific applications (i.e., cyber-physical systems, decision sciences, and data products). This way, both Faria and Varshney et al. presented the first steps on defining safety concepts for machine learning applications. Later, other authors focused on these specific machine learning safety concepts, such as

classifying the major sources of uncertainty, and how to treat it during development and testing of ML models [22].

In parallel, other authors have conducted their research on the verification and validation of safety-critical ML-based systems, with a special focus on the certification challenges, and on the gaps of the currentInternational Organization for Standardization (ISO) standards for the automotive domain [10,23–26]. These authors described adaptations that can be considered critically needed to allow these systems to be compliant with ISO 26262, also identifying suggestions on how the standards should evolve. More recently, and still in the context of autonomous driving, Schwalbe and Schels [27] published a survey on methods for safety assurance of ML-based systems, where a structured certification-oriented overview on available methods supporting the safety argumentation of ML was provided.

At the same time, researchers were also concerned with machine learning engineering from a software quality assurance perspective. Ma et al. [28] presented an extensive literature review and proposed a Secure Deep Learning Engineering (SDLE) development process specialized for Deep Learning (DL) software. This topic also got the attention of other researchers who focus on safety architectural design and architectures for intelligent systems and safe ML [13], and also on the low-level requirements of machine learning models code, by proposing requirements traceability of software artifacts [29]. In a more recent study, Jenn et al. [18] presented the main challenges on the certification of systems embedding ML.

One of the surveys published recently also addresses the safety of ML following an approach very similar to ours. Ashmore et al. [16] presented a theoretical framework of safety considerations in an ML component development lifecycle, detailing specific assurance properties for each phase activity, and an overview of the state-of-the-art methods to provide the required assurance evidence. The two works are very close to each other because they both present the safety concerns along the ML lifecycle phases; however, the work of Ashomre et al. is concerned with obtaining evidence for providing assurance to ML components. In contrast, our work is mainly focused on exploring the susceptibilities of each lifecycle phase activity where safety hazards could be introduced.

All the literature presented in this section constituted a fundamental basis for constructing our paper. Nevertheless, since we do not focus on software architecture or on assurance patterns, we consider that our work positions itself in a different perspective compared to the remaining state-of-the-art papers on safety of machine learning: by presenting a clear and structured analysis of safety hazards detailed according to phases and activities of the ML lifecycle, and by proving for every identified hazard an illustrating example based on a real-world application scenario.

3. Safety and Machine Learning

Machine learning algorithms are increasingly used in systems that involve humans or operate in shared environments with humans, resulting in growing attention to the aspects of the safety of those systems in the last few years. To provide a better understanding of the terms and notation of safety and machine learning fields, this section presents clarifications for both areas individually, and then for the intersection of both. For that purpose, the next subsections will focus on the definition of a few statistical machine learning notations, as well as on the relationship of those notations with safety concepts. At the end of this section, safety in machine learning is further illustrated by presenting an example application of a safety-critical CPS where an ML model is integrated. The example presented here will also be considered in Section 4.

3.1. Safety

Across multiple engineering disciplines, safety is a frequently used term, which designates the absence of catastrophic consequences on the user(s) and on the environment [30].

The concepts of **harm**, **risk**, **hazard**, and **uncertainty** are inherent to safety. Without focusing on machine learning, a system generally returns an outcome based on its state and on its inputs.

This outcome may be desired or not, and an undesired outcome is only harmful if its consequences exceed some threshold that is quantified by society. Therefore, **harm** is defined as a physical injury or damage to the health of people, either directly or indirectly as a result of the damage of a property or to the environment. **Risk** is the product between the probability and severity of a potential harm, and a **hazard** is defined as a potential source of harm [31]. Harmful outcomes occur in unexpected or undetermined scenarios and operating conditions. **Uncertainty** is connected with safety when the outcome of a system is unknown and its probability distribution is also not known (or only partially known) [32]. **Epistemic uncertainty** refers to uncertainty caused by a lack of knowledge of the physical world (knowledge uncertainty).

A general definition of safety is related to the minimization of **risk** and **epistemic uncertainty**, in order to minimize unwanted outcomes severe enough to be qualified as harmful [33]. Engineering safe systems relies on minimizing the **risk**-product, either by minimizing the probability of the harmful events or by reducing their damage. In order to achieve this, potential hazards must be carefully identified and their risk assessed, strategies to eliminate or mitigate the identified hazards must be implemented [34]. In addition, in most industries, the proper implementation of safety strategies must be documented in safety cases, providing a convincing argument that the system is fulfilling its safety goals for a given application in a given environment [35].

The success of the safety analysis fully depends on the complete and accurate knowledge of multiple aspects of the system, such as the operating environment, system requirements, design, and constraints. Thus, a safety assessment intrinsically has some level of **epistemic uncertainty**. Keeping these concepts as background, the next step will be to analyze them as applied to the field of machine learning.

3.2. Machine Learning and Notations

Machine learning is the science of programming computers that enables them to learn from data [36]. It is a sub-field of computer science that evolved from the study of pattern recognition and computational learning theory, exploring the construction of algorithms that can learn and make predictions based on data [37]. Such algorithms operate by training a model from data inputs in order to make data-driven predictions or decisions, instead of being hand-crafted to solve a particular problem by means of "conventional" algorithms.

ML systems can be classified according to the amount and type of supervision they get during learning, there exists four major categories: Supervised learning, Unsupervised learning, Semi-supervised learning, and Reinforcement learning [36]. On Supervised learning, the data used for the learning process includes the desired solutions (i.e., labels/annotations); on the contrary, for Unsupervised learning the data are unlabeled (the system tries to learn without a teacher). Within Supervised learning, problems can be divided into classification and regression, where classification models rely on identifying a class (i.e., label) that the input belongs to, and regression models predict continuous-valued attributes. For Unsupervised learning, models are based on cluster algorithms that are used to group similar inputs into sets. In Semi-supervised learning, some algorithms can deal with partially labeled data (usually a lot of unlabeled data and a small amount of labeled data). Lastly, Reinforcement learning is based on a different approach, where the learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards) [38].

Machine learning builds on the foundations of Statistical learning theory. James et al. [39] refer to Statistical learning as a vast set of tools for understanding data. The starting point in the theory and practice of Statistical machine learning is **risk minimization**. Given the random example space $X \times Y$ of observations (features) x and labels y, with a probability density function $P(X, Y)$, where the goal is to try to find a function $f : X \rightarrow Y$ that approximates the most probable label y for the observation x. Considering the **loss function** L as the measure of the error incurred by predicting the label of an

observation x as $f(x)$ instead of y, Statistical learning defines the **risk** of f as the expected value of the **loss** of f under P:

$$R(f) = \int L(x, f(x), y) dP(x, y) \tag{1}$$

Therefore, the Statistical learning problem consists of finding the function f that optimizes (minimizes) the risk R. However, in an ML context, the probability function $P(X, Y)$ is not known, only a finite number of m examples $(x^{(i)}, y^{(i)})_{i=1...m}$ are available. This way, for the algorithm hypothesis h and the loss function L, the expected loss on the training set is defined as **empirical risk** of h:

$$R_{emp}(h) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, h(x^{(i)}), y^{(i)}) \tag{2}$$

Another important aspect is the ML model performance evaluation, which is the process of understanding how well a model gives the correct prediction. In some cases, it is important to consider how often a model gets a prediction correct, however in other cases, it is important to look if the model gets a certain type of prediction correct more often than the others [40]. There are several metrics that we can use to evaluate the performance of ML algorithms, which are dependent on the type of the problem (e.g., classification, regression, clustering). For classification problems, the predictions could be classified as: (1) True Positive (TP), where both prediction and label are positive; (2) False Positive (FP), where the prediction is positive but the label is negative; (3) True Negative (TN), where both prediction and label are negative; (4) False Negative (FN), where the prediction is negative but the label is positive. Different evaluation metrics can be calculated through the combination of these four values, being accuracy, precision, and recall the most used ones.

3.3. Safety in Machine Learning

The definition of machine learning safety has not yet been formalized among the community. Several authors define machine learning safety in terms of **risk**, **epistemic uncertainty** and the **harm** caused by unwanted outcomes [11,20,21].

Based on this generic notion, various approaches towards safety in machine learning focus on reducing the likeliness of undesired outcomes and the induced **harm** by explicitly including these aspects into the **loss function** L, thereby optimizing the machine learning system towards achieving a safe behavior. Whilst this approach may form one of the core safety strategies to be used in the context of machine learning, it falls considerably short of being sufficient to achieve safety as required in real-world applications such as autonomous driving or factory automation in environments shared between humans and machines.

In such contexts, we usually have to guarantee that the probability of harmful behavior is extremely low. Applicable functional safety standards such as ISO 26262, IEC 61508, and SAE ARP4754A, require the demonstration of failure probabilities of, for example, 10^{-7} and 10^{-9} per operating hour [41–43]. The **loss function** approach as a single separate means is not capable of achieving these levels. In addition to this shortcoming, the loss function approach does not adequately address issues related to **epistemic uncertainty** and the proper formulation of the loss function itself (these will be analyzed in detail throughout Section 4).

Further, the **loss function** will typically combine different aspects to define inferior behavior and undesired outcomes. Many outcomes of machine learning systems may be identified as undesired by such combined loss function, but may be tolerable from a safety perspective. This is due to the fact that these outcomes either do not lead to harmful behavior, or they are detectable, and therefore can be masked by the embedding CPS. Making the distinction between an acceptable and a potentially harmful undesired outcome is one of the challenges of applying ML-based control systems in safety-critical environments.

A related challenge is to define an adequate safety-directed reaction for those undesired outcomes that are identified as potentially harmful. In many current and future safety-critical applications, such as fully autonomous driving without a safety driver on board, there is no safe state in the traditional sense. To ensure the proper functioning of the ML-based control system within its embedding CPS, the failure modes of the ML-based control system must be understood as well as the consequences of these failure modes may have at the level of the embedding CPS. Consequently, safety engineering for CPS relying on ML-based control needs to cover both, the control system based on machine learning, and also the embedding CPS as the operational context.

To properly address the safety of machine learning systems, one has to implement the rigorous processes prescribed in functional safety, and utilize both: safety strategies known from conventional systems development, and safety strategies developed specifically for machine learning systems. The challenge here is to bridge the gap between the traditional defensive, risk-averse, and rigorous functional safety culture, and the opportunity-oriented, technology-focused culture still dominating in machine learning. In this paper, we aim to contribute to connecting these seemingly antagonistic views by identifying the hazards and related failure mechanisms that may potentially lead to unsafe behavior of machine-learning-controlled CPS, thereby providing a stable basis for identifying applicable safety strategies addressing each one of these hazards.

3.4. Example Application

As mentioned above, machine learning algorithms are affected by probabilistic factors inherent to their design, which consequently leads to several concerns that need to be assured for the guarantee of safety in high-critical applications. With the goal in mind of presenting the whole process of designing an ML-based system, we describe an example application that will be used as a mental reference to provide guidance throughout the paper.

Not surprisingly, industries around the world are exploring the possibility of applying ML in their operations [44]. Focusing on the example of the factories of the future, various Artificial Intelligence (AI) use cases involve improving productivity across major areas of operations, both outside and inside the factory walls. Inside the factory walls, AI can provide various benefits to production, and to such support functions as maintenance and logistics. Keeping this in mind, we select an example based on a self-driving vehicle that operates in factories and warehouses sharing transportation routes and tasks with human workers. Recent research shows an increasing interest in the use of machine learning algorithms for self driving vehicles inside factories [45]. The inevitable involvement of human operators in their supervision and control introduces significant challenges in assuring safety while achieving the expected performance.

Overall, autonomous vehicles rely on diverse technologies: sensors, actuators, complex algorithms, and powerful processors. ML components are developed for scene recognition, path planning, and vehicle control tasks. It is important to mention that we will only focus on the ML-based scene recognition component for further exemplifications. From a high-level perspective, vehicles create a map of their surroundings, by monitoring the position of nearby objects (other vehicles, people, signs, etc.). These complex algorithms are usually composed of ML models that operate in scene recognition tasks, which are based on localization and object detection and tracking [46]. This process is depicted through a high-level diagram in Figure 1. For safe movement of the self-driving vehicle inside factories, all of the system components must work together in order to avoid hazards.

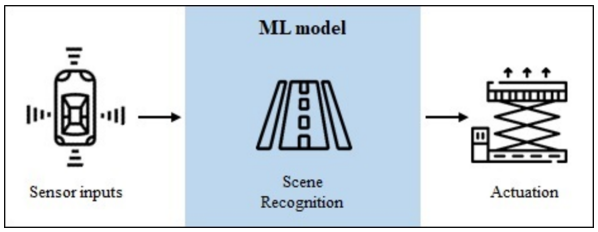

Figure 1. Simplified diagram of an autonomous vehicle operation during run time. A Machine Learning (ML) component is responsible for scene recognition tasks.

The example presented here focuses on an autonomous vehicle, controlled by an ML model, which transports material between assembly cells, using data from cameras and sensors to avoid obstacles. Considering a safety-critical perspective, this example illustrates the case where humans and robots (i.e., self-driving vehicles) work together, and the safety of the human being must be guaranteed. Therefore, the scenario of collision represents a critical harm where the vehicle may collide with obstacles (humans) due to failures in one of the components mentioned. Collisions can lead to fatal injuries, which are not avoidable without safety measures.

4. Machine Learning Lifecycle

In safety-critical systems it is important to understand how these systems work and interact with the surrounding environment, and respond to changes. Considering the application of a machine learning model in one of these systems, assuring its safe operation requires a complete understanding of the machine learning lifecycle, in order to identify in which step safety could be compromised by development related safety hazards.

The ML lifecycle represents the process of the development and integration of a machine learning algorithm into a system. Similar to traditional system development, this process is constructed based on a set of system-level requirements, from which the requirements and operating constraints for the ML models are derived. Nonetheless, machine learning development algorithms require dealing with new paradigms that are not common in the standard software development lifecycle. Instead of a well-defined set of product features to be built, as in traditional software, ML development involves the acquisition of new datasets and is based largely on experimentation: different models, software libraries, tuning parameters, etc. The manipulation of the datasets and the use of ML training techniques, produces a pipeline that takes data as input and delivers trained models as output, for later integration into systems. The successful development and integration of an ML model happens when the system is capable of achieving the specified requirements for new unseen data [47].

Commonly, the ML lifecycle is divided into five main phases: **Requirements**, **Data Management**, **Model Development**, **Model Testing and Verification**, and **Model Deployment**, which are depicted in Figure 2. Here, we considered that the **Requirements** phase outputs the requirements list for the remaining four phases, which we encapsulate in the block **Design and Development**. This block compromises every phase necessary to output a deployed ML model into a system (i.e., self driving vehicle). This section describes each phase in detail, presenting the activities associated, and the potential hazards that could be introduced. Additionally, the example of the self-driving vehicle inside factories (described in Section 3.4) will be considered here, presenting in each lifecycle phase an illustrative case of the potential hazards and how they could affect system performance.

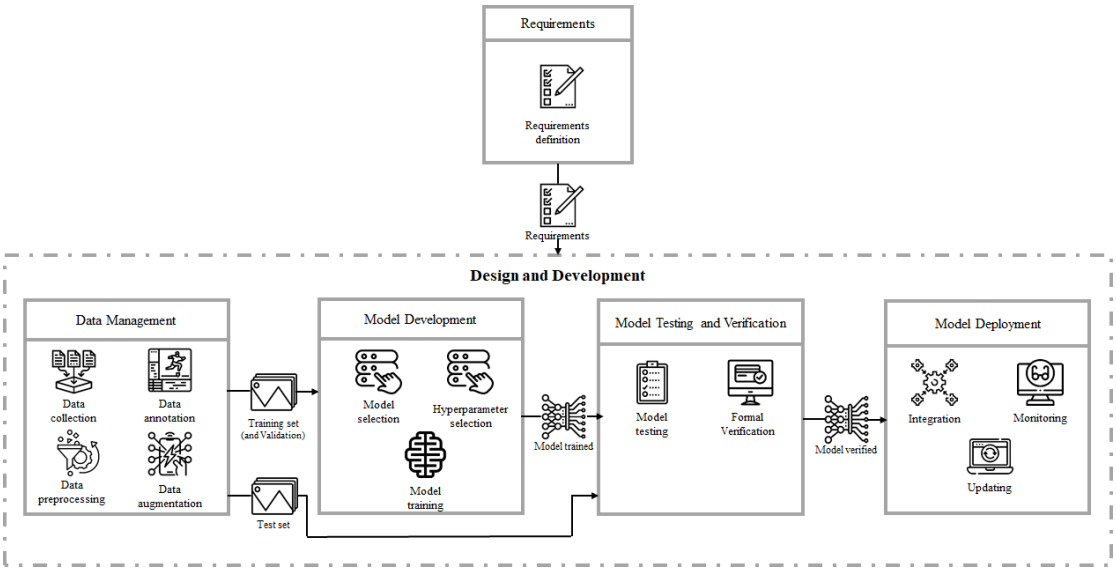

Figure 2. Machine learning lifecycle (adapted from the work of Ashmore et al. [16]).

4.1. Requirements

The main goal of the **Requirements** phase is to define the right specifications for the intended use of a system. Recently, authors have been discussing the challenges of requirements engineering for ML-based systems, suggesting that ML engineering constitutes a paradigm shift compared to conventional software engineering [48], and that requirements engineering is the most difficult activity for the development of ML-based systems [49].

The **Requirements** phase identifies the functionality, performance levels, and other characteristics that the system must satisfy in order to be acceptable to the customer/final operation. During requirements engineering, the system architecture is constructed up to the smallest functional units, and for each, the use case is specified, including intended operational environment and behavior [27]. Requirements can be functional (FR) or non-functional (NFR), where typically, FRs describe the system functionality, while NFRs describe system properties and constraints [50] (e.g., an FR for the self-driving vehicle example could be *the system has to detect obstacles*, and an NFR could be *the detection speed of the system should be less than 5 s*). Furthermore, requirements are often organized hierarchically, where high level requirements focus on what should be achieved, and lower ones on how to achieve it. In traditional software engineering, there is a direct link (traceability, that is, proof of the complete implementation of requirements) between low and high level requirements, as well as design (the total set of low level requirements must be validated against the higher level parent ones). However, in an ML-based system, this traceability is not straightforward, since it is not based on an explicit implementation of requirements for achieving a desired function, but instead a generic calculation algorithm (e.g., a neural network), the behavior of which is defined by parameters (e.g., weights). For that reason, high level requirements can be explicitly defined but the low level ones are given implicitly when specifying the dataset (the characteristics of which are defined in the **Requirements** phase and are implemented during the **Data Management** phase). Without explicit low level requirements, the traditional traceability-based assurance approach—linking high-level to low-level requirements, and low-level requirements to implementation—cannot be applied. On the other hand, even if there would be explicit low-level requirements it would be hard to trace them into the implementation, telling which elements of the machine learning algorithm and which of its parameters contribute to implementing a specific low level requirement. Consequently, the traditional way of assuring software functionality cannot be applied [51].

At the end of this phase, the set of requirements defined is provided to the **Design and Development** phases.

4.1.1. Activities

For the **Requirements** phase we only considered the activity of *requirements definition*, which focuses on the definition of several conditions that need to be satisfied along the rest of the ML lifecycle. For that reason, the following requirements need to be considered during:

- **Data Management**: Requirements focused on defining the desired dataset according to the use case. Identification of all possibly relevant sources of data that may help the ML system to provide accurate and robust results. The definition of the data will impact the dataset distribution and the ML model effectiveness to generalize.
- **Model Development**: Requirements focused on the objective/loss function and performance measures definition. Definition and discussion of the performance measures and expectations by which the ML system shall be assessed. As mentioned by Vogelsang et al. [48], performance measures such as accuracy, precision, or recall are not well understood by customers. On the other hand, the adequacy of the measures depends on the problem domain. Therefore, requirements should consider the demands of the stakeholders by translating them to the appropriate measures.
- **Model Testing and Verification**: Definition of testing and verification requirements related with performance metrics, fault tolerance, and scenario coverage. Here it is important to clarify which scenarios should be tested, which performance metrics should be achieved, and which faults the systems should tolerate.
- **Model Deployment**: Specifications of conditions for data anomalies that may potentially lead to unreasonable or non-safe behavior of the ML system during runtime. The specification should also contain statements about the expected predictive power: performance on the training data can be specified as expected performance that can immediately be checked after the training process, whereas the performance at runtime (i.e., during operations) can only be expressed as desired performance that must be assessed during operation.

4.1.2. Hazard Identification

The **Requirements** phase is crucial for ensuring that a model is fitted to its purpose within a specific context. As mentioned above, *requirements definition* affect the remaining ML lifecycle phases; therefore, the hazards introduced during this phase will have a direct impact on them.

- **Incomplete definition of data**. When defining the operation mode of an ML-based system it is important to have a complete definition of all data examples in order to avoid future problems on the dataset distribution, which will surface during the **Data Management** phase. This assumes that humans can correctly identify all varying conditions at requirement specification time, which is a difficult mark to achieve.
- **Incorrect objective function definition**. One of the requirements that needs to be defined is the objective function, that is, the loss function that needs to be minimized. The definition of this function could cause different problems in case the wrong objective function is defined, or that some undesirable behavior occurs during the learning process. The incorrect definition of the objective function could potentially harm any ML-based system, since its maximization leads to harmful results, even in the limit of perfect learning and infinite data. This could happen when the designer/requirements engineer specifies an objective function that focuses on accomplishing some specific task in the potentially very large environment, but ignores other aspects of it. By doing that, it implicitly expresses indifference over environmental variables that might actually be harmful to change, resulting in negative side effects specifically if human safety is not correctly accounted for.
- **Inadequate performance measure**. The definition of the performance measures happens during this phase and it will have a direct impact on model training during the **Model Development** phase and ultimately on the delivered model. Although there exist several measures that can be used for evaluating the performance of ML algorithms, most of these measures focus mainly on

some properties of interest in the domains where they were developed. According to a specific system, different predictions can be of great importance in safety-critical systems, and deciding which measure we want to optimize will influence the system behavior.

- **Incompleteness on testing/verification**. The definition of the entire necessary testing and verification scenarios may not be complete, and/or values to be achievable for different metrics may not be correctly defined. For both cases, incompleteness in the quantity and quality (i.e., threshold values for metrics) could lead to safety hazards (e.g., error rate, execution time).
- **Inadequate safe operating values**. During the **Model Deployment** phase, the monitoring of outputs requires the definition of some kind of "measure of confidence" and/or a "comparison to reasonable values" in order to detect incorrect outputs. The inadequate definition and/or implementation of this requirement could potentially provoke a safety hazard.

Considering our example application of a self-driving vehicle in a factory, the **Requirements** phase can introduce several hazards as mentioned above. At the very beginning, when defining the dataset examples, we should consider under which conditions the self-driving vehicle will operate (e.g., amount of light, identification of which objects/obstacles it should detect). If this definition is not performed adequately, this could result in an **incomplete definition of data**. For example, if the amount of light in the room is less than some lumens value, and this scenario was not defined in the requirements, the vehicle could miss the detection of an object and cause harm.

On the other hand, an **incorrect objective function definition**, may occur for the case where the importance of detecting certain obstacles it is not balanced correctly (e.g., the error of not detecting a person should have a higher weight than the case of not detecting a box).

An **inadequate performance measure** selected that could also impact the operation of the self-driving vehicle. For example, a system that is supposed to detect humans and other obstacles, fails to classify human presence, resulting in an FN case and ultimately in crashing into a person. On the other hand, in an FP case (i.e., false alarm), such an ML-based detection system may result in automatically applying the brakes of the vehicle to prevent crashing into what the algorithm identifies as an obstacle. Minimizing FN seems to have more weight than minimizing FP, because in this scenario FP could be considered to have only an effect on operational performance. However, such FP could also cause a safety-critical hazard if the vehicle stops for no necessary reason. Therefore, existing measures need to be carefully optimized in order to fit the needs for evaluating a safe performance of ML models [52].

The **incompleteness of validation/verification** requirements also represents a safety hazard in the factory context when the definition of some value is incorrect or incomplete. Defining a non-sufficient runtime performance can later translate into hitting an obstacle, if for example, an obstacle was detected within the value defined; however, since the definition was incorrect, it still caused an accident.

Lastly, in autonomous vehicles the term "safe state" refers to a state where the risk from the system is reasonable [53]. However, when using the term safe state, the challenge relies on the identification of a threshold, that is, an **inadequate safe operating value** under which the risk level is not acceptable. Therefore, the definition of this value could also result in an accident in the factory.

4.2. Data Management

The **Data Management** phase is responsible for acquiring meaningful data for the machine learning model to be able to make predictions on future data. The inputs of this stage are the set of requirements that the application has to achieve, and the outputs are composed by the training and test datasets, to serve as inputs for the **Model Development** and **Model Testing and Verification** phases, respectively. For producing such datasets, this phase focuses on four different activities: data collection, data annotation, data preprocessing, and data augmentation.

4.2.1. Activities

The goal of data collection is to find datasets that can be used to train ML models, gathering data samples representative of the real-world system or process for which an ML model needs to operate. These data can be obtained by data discovery and/or data generation [54]. Data discovery is necessary to share or search for new datasets, and has become important as more datasets are available on the Web and corporate data lakes. Despite the value of data, the machine learning community has many public datasets in different areas [55,56]. On the other hand, data generation can be used when there is no available external dataset, but it is possible to generate manual or automatic datasets instead. For manual construction, crowd-sourcing is the standard method where human workers are given tasks to gather the necessary bits of data that collectively become the generated dataset. Alternatively, automatic techniques can be used to generate synthetic datasets [57]. For mostly industrial applications it is unlikely that a specific dataset already is available; therefore, all of the aforementioned techniques are usually explored.

Alongside data collection is data annotation, which is the human process of labeling data available in various formats (e.g., text, video, images). Data need to be precisely annotated using the right tools and techniques. Having annotated datasets improves the model accuracy at the end because we have more variety of datasets to be used later in the training phase.

After collecting the data, data preprocessing is the step to produce consistent datasets, seeking to reduce complexity of collected data or to engineer features to support training. Real-world data may be incomplete, noisy, and inconsistent, which can disguise useful patterns. This is due to (1) incomplete data: lacking of attribute values or certain attributes of interest, or containing only aggregate data; (2) noisy data: containing errors or outliers; (3) inconsistent data: containing discrepancies in codes or names [58]. Data preprocessing generates a dataset smaller than the original one, which can significantly improve the performance of an ML model. This task includes selecting relevant data, through attribute selection using filtering and wrapper methods, and also removing anomalies, or eliminating duplicate records. For unbalanced data, over or under sampling approaches help to deal with imbalanced datasets. Additionally, some data exploration could be performed in order to find correlations, general trends, and outliers.

Lastly, data augmentation complements previous steps where existing datasets are enhanced by adding more external data and small perturbations to the real data. Creating synthetic examples helps the model to generalize, and be invariant to those perturbations (e.g., noise, rotation, illumination). In the data management community, entity augmentation techniques have been proposed to further enrich existing entity information [54].

4.2.2. Hazard Identification

Data are the core of machine learning. An ML algorithm makes predictions based on a model calculated from its input data, and its performance is usually dependent on the quality of these data. The activities during the **Data Management** phase could be susceptible to the introduction of some error that would potentially cause a misbehavior in the future. Among the many challenges in machine learning, data collection is becoming one of the critical bottlenecks. Except for use cases where data already exists in a preprocessed form (e.g., Fashion-MNIST [55] and COCO [59] datasets), the majority of the time for running machine learning end-to-end is spent on preparing the data. Several dimensions of concern that are relevant for safety can be identified during this step:

- **Inadequate distribution**. Real-world data are usually imbalanced and it is one of the main causes for the decrease of generalization in ML algorithms [60]. As already mentioned in the **Requirements** phase, distribution problems may occur when the definition of the requirements for the dataset are not complete. Several other aspects of the distributions can contribute to a safety hazard. First, if the distribution of the training examples does not adequately represent the probability distribution of (X, Y) (e.g., due to lost or corrupted data), the learned hypothesis H may be susceptible to

failure [11]. Additionally, the distribution of input variables can also be different between training and the operating environment. This issue is called covariate/distribution shift, which is one form of **epistemic uncertainty** [21]. ML algorithms assume that training and operating data are drawn from the same distribution, making algorithms sensitive even to small distribution drifts. Moreover, datasets are not complete when rare examples are absent or under-represented due to their small probability density (they were probably not anticipated when acquiring data), which could yield a substantial mishap risk [11,20]. Here, rare examples could be divided into corner cases (i.e., infrequent combinations) and edge cases (i.e., combinations that behave unexpectedly, so the system does not perform as required/expected) [61]. For these cases, the learned function h will be completely dependent on an inductive bias encoded through H rather than the uncertain true distribution, which could introduce a safety hazard. Additionally, edge cases also incorporate adversarial attacks, where deliberated actions are taken by an adversary, which manages to modify input data in order to provoke incorrect outputs and harm the system and the surroundings [62]. Despite adversarial attacks happening only during the **Model Deployment** phase, not considering adversarial inputs as part of the dataset could potentially cause some harm later on in the lifecycle. For that reason, this hazard is being presented here, since the dataset acquired should account for input examples of possible adversarial attacks in order to prevent a future safety hazard. Furthermore, other sources that affect distribution rely on datasets that were obtained synthetically, which could impact distribution by not accounting for real-world scenarios, or by introducing some bias if they generate inputs with frequent features (e.g., all generated examples containing the same background color or noise). All the hazards presented above are mostly connected with data collection activity. Other possible hazards affecting distribution are introduced during data preprocessing, for the case where the partition of the dataset into train, validation, and test set is not done correctly and affects distribution.

- **Insufficient dataset size**. First, as ML is used in new applications, it is common that there is not enough training data. Traditional applications, like machine translation or object detection, have massive amounts of training data that have been accumulated over decades. On the other hand, more recent applications have little or no training data. It takes a lot of data for most ML algorithms to work properly and generalize well on unseen data. Even for very simple problems, thousands of examples could be needed, and for complex problems such as image or speech recognition, millions of examples may be required [36]. Size is connected with distribution, since having a small dataset directly influences the distribution and total coverage of all possible cases. The dataset has to have a minimum size in order to be representative and adequately distributed. ML models can overfit if the dataset is too small (i.e., the case of high variance, where variance is defined as the difference in performance on the training set vs. the test set). Even if a dataset is adequately distributed if it is too small, the model has only a small number of cases to learn from, which will affect its general performance and potentially cause a safety hazard.
- **Bias**. There are different types of bias that could be introduced when data are collected to train and test a model. Considering the dataset distribution mentioned above, when examples are missing in a dataset, we are introducing *sample bias*. Here, the variety and amount of samples collected will introduce bias on the model if the samples are not representative. Data examples could also come from different sources, so we should also keep in mind that if all the samples of training and/or testing sets come from the same source, we could be introducing *measurement bias*. Other types of bias connected with distribution is *confirmation bias* [63]. This type of bias happens when we focus on information that confirms already held perceptions. That is, samples might have some features that appear together on the collected data but their connection has no meaning in the real-world. This way, ML models could learn incorrect features to make a prediction because they seem to be correlated for some random reason (i.e., they keep showing up together on samples). Additionally, during data prepossessing, some important samples and features could be removed, which contributes to *exclusion bias*.
- **Irrelevance**. The data acquired contains extraneous and irrelevant information, ML algorithms may produce less accurate and understandable results, or may fail to discover anything of use at

all. Relevance here considers the intersection between the dataset and the desired behavior in the intended operational domain [16].

- **Quality deficiencies**. Each activity of the **Data Management** phase affects data quality. First, during data collection, all kinds of data collected (based on sensors but also on human input) are limited in their accuracy and can be potentially affected by various types of quality issues. This property can also be defined as accuracy, since it considers how measurement (and measurement-like) issues can affect the way that samples reflect the intended operational domain, covering sensor accuracy [16]. Next, during data annotation, quality could be compromised due to the incorporation of incorrect labels or incorrectly annotated area, since it is a task mainly performed by humans. During preprocessing, different techniques such as rescaling, filtering, and normalization contribute to a delta between the quality of the cleaned data and the data on which the model is currently being applied, which contributes to **uncertainty** [22]. Lastly, during data augmentation, the inclusion of non-realistic examples could affect dataset quality by using augmentation techniques, which generate data that do not make sense and change the complete meaning in a sample.

ML models learn from a subset of possible scenarios represented on the training dataset, which determines the training space, defining the upper limit of the ML model performance [64]. As pointed out above, the feature space could miss data, be incomplete, and can cover only a very small part of the entire feature space, with no guarantee that the training data are even representative [23].

Considering the example of a self-driving vehicle in a factory, the **Data Management** phase plays a major role, especially due to **inadequate distribution**. Since the input distribution cannot be controlled, which compromises the assumption that the training samples were drawn from the same underlying distribution as the testing samples [13]. As mentioned above, data could be lost or corrupted or even acquired from different sources, due to the fact that sensors were replaced between acquisition and operating phases (e.g., different models, different specifications), or even the case that the routes acquired are from different shop floors between training and operating phases. Both examples cause a mishap between training and testing distributions (distribution shift), and are closely connected to sample and measurement bias. The completeness of the dataset regarding rare examples relies on the under-representation of corner and edge cases. Some examples of corner cases could be described by the small probability of some object being present on the factory shop floor (e.g., new robots, machines), or even by the case of some human worker being completely or partially covered by an object (occlusion problem), and not being detected. On the other hand, the unexpected behavior of an edge case, could be that an unusual obstacle is actually on the shop floor (something we cannot think of), or even, for example, some strange behavior of the ML model due to a person being dressed with stripes (that could confuse the model to detect a lane marker). Still on the edge cases, but considering the scenario of the factory suffering an attack, an adversary might slightly modify an image to cause it to be recognized incorrectly, and provoke a non-safe action by the vehicle (for example, some "STOP" or "DANGER" sign placed on the factory could be modified by an adversary, preventing the self-driving vehicle from stopping and making it possible to cause an accident—here we are only considering modifications of an object's appearance that do not interfere with a human observer's ability to recognize the object).

Furthermore, the problem of **insufficient dataset size** is related to the case of not enough acquired data, preventing the model from being able to generalize (e.g., some obstacle detection with low accuracy prevents the vehicle from stopping and provokes an accident).

In terms of **bias**, and in specific the *confirmation bias*, one example could be described by the case where in every sample of the dataset, there is a human worker and some other feature that by coincidence (e.g., a specific machine, a specific background color) always appear together. This way, the model could potentially learn that there is a person in every sample where this specific feature is present (or vice-versa). In that case, the model is biased and can cause harm.

Data could be **irrelevant** for different reasons: if the training dataset only contains one floor of the warehouse and not the complete building, or if a self-driving vehicle is reallocated into another building of a different type of factory, and the data acquired will no longer be completely relevant.

Lastly, data **quality inefficiencies** could emerge during all activities. During data acquisition, if a problem on data sources (e.g., cameras, radars, LiDARs, ultrasonic sensors, GPS units and/or inertial sensors) occurs compromising its quality, and producing noise and unreliable data (e.g., acquiring image data with some defect on the camera will then compromise, for example, the detection of a moving object). Also during data annotation, quality is affected, since it is mainly performed by humans. Errors such as the incorrect identification of the area that some object (e.g., human, machine) occupies in an image, and the attribution of the wrong label (e.g., machine instead of human) could cause harm. The quality of data could also be affected during preprocessing, if for example, images acquired by the cameras of the self-driving vehicle are rescaled for computation reasons (this specific case could not lead to harmful results; however, it could have a potential effect on model performance). During the data augmentation, an image rotation or a noise added to the inertial sensor signal, could not be translated into a real sample in the operation scenario, affecting data quality.

4.3. Model Development

This phase is responsible for training a model based on the training dataset given by the **Data Management** phase, reproducing the correct relationship between inputs and outputs. The trained model will then be given to the next phase for its verification and testing. The **Model Development** phase is composed of a set of decisions related to models and parameters, and also by the training part of the ML model development process.

4.3.1. Activities

Given a prediction task, the first step is model selection. This process starts from analyzing well-known models that have been successful in similar task domains. This activity decides the type of model according to the problem that needs to be solved. As mentioned before, different machine learning models exist for different problems, for example, classification, regression, and cluster analysis.

In ML, we have both model parameters and parameters that are later tuned during model training. These tuning parameters are called hyperparameters, and they deal with controlling optimization function and model selection during training with the learning algorithm. The hyperparameter selection focuses on ensuring that the model neither underfits nor overfits the training dataset, while learning the structure of the data as quickly as possible. The subset of data used to guide the selection of hyperparameters is called the validation set [65].

After model and hyperparameter selection, the model needs to be trained on the training data provided by the **Data Management** phase. This way, during tmodel training, the model is able to understand patterns, rules, and features of the training data, optimizing the performance of the ML model with respect to an objective function, which specifies the model requirements (defined in the **Requirements** phase, in Section 4.1). Here, the training data given are divided into training and validation sets. The training set is used to find internal model parameters (such as weights of a neural network, or polynomial coefficients) which minimize an error metric previously selected (also in the **Requirements** phase). The validation set is then used to assess the model performance. These two steps are iterative in order to tune not only the model parameters but also the hyperparameters of the training task. Although there is no consensus on the best methodology for tuning hyperparameters, the most common techniques include initialization with values offered by ML frameworks, manual configuration based on recommendations from literature or experience, or trial and error [16]. Due to the lack of understanding of the exact nature between the parameters learned during model training and the performance of the trained model [66], the adjustments and tuning inside the loop are driven by heuristics (i.e., adjusting hyperparameters that appear to have a significant impact on the learned weights). Thus, many similar models are trained and compared, and a series of model variants needs

to be explored and developed. Due to the expensive training phase, each iteration of the modeling loop takes a long period of time and should produce many (checkpointed) snapshots of the model for later analysis [67].

4.3.2. Hazards Identification

One of the main concerns regarding ML models is the software implementation. Generally, a programmed component is one that is implemented using a programming language, regardless of whether the programming was done manually or automatically [23]. On the contrary, an ML model is one that is a trained model using a supervised, unsupervised, or reinforcement learning approach. The probabilistic nature of the training phase and the amount of parameters and hyperparameters to be defined, leads to several potential hazards that could compromise safety:

- **Model mismatch**. During the model selection activity, the selected ML model or the model architecture does not fit the use case application, not fully covering the requirements defined. The choice of a particular model depends on the size of the training data, number of relevant features, linearity (if classes can be separated by a straight line), the goal for an interpretable or accurate model (trade-off between accuracy and interpretability), and the time and resources for training [68]. Additionally, model selection could also be affected by the computational power required to train a model. To reduce computational costs, the complexity of the models selected is restricted, which commonly leads to a decrease in model performance. Therefore, in some cases a trade-off may be required when the computational power is limited to some extent [16], which is related to the model mismatch hazard.
- **Bias**. The bias introduced in this phase is called *algorithm bias*, and it comes from the model selection activity. This bias is a mathematical property of an algorithm.
- **(Hyper) Parameters mismatch**. The methodology for selection and initialization of the model parameters or the training hyperparameters has a high impact on model performance. This happens because the same training process can produce different training results, since there are several techniques for model weights initialization and hyperparameter tuning.
- **Performance measure mismatch**. The selection of the adequate performance measures happens during the **Requirements** phase, as mentioned in Section 4.1; however, the impact of its definition takes effect during the **Model Development** phase, specifically during model training. If there is a mismatch between the selected performance measure(s) and the requirements, the ML model could be inadequately optimized during training.
- **Error rate**. Although an estimate of the true error rate is an output of the ML development process, there is only a statistical guarantee about the reliability of this estimate. Even if the estimation of the true error rate was accurate, it may not reflect the error rate that the system actually experiences while in operation after a finite set of inputs, since the true error is based on an infinite set of samples [23]. Additionally, the probability of failing is intrinsic to an ML model. The system is not able to ensure the complete correctness of an ML module output in the user environment where unexpected input occurs sporadically. Furthermore, wrong predictions could happen; however, there is not an impact in the system behavior (model "silently" fails) compromising the definition of safety-critical scenarios, which could later become a problem.
- **Lack of interpretability**. The process of integrating machines and algorithms into peoples daily lives requires inseparability to increase social acceptance [69]. Interpretability in ML supports assurance [16] since it provides evidence for: (1) justifying results (i.e., explanation of a decision for a specific outcome, particularly when unexpected decisions are made. and also justifications in order to be compliant with legislation); (2) preventing things from going wrong and identifying and correcting errors (i.e., the understanding about the system behavior provides greater visibility over unknown vulnerabilities and flaws); (3) assisting model improvement (i.e., a model that can be explained and understood is one model that can be more easily improved); and (4) supporting the understanding of the operational domain (i.e., a helpful tool to learn new

facts, gather information, and thus to gain knowledge) [69]. ML and DL models can have millions or billions of parameters, and the most successful constructs are very complex and difficult to understand or explain [70]. Interpretability does not ensure safety by itself, but can help to understand where models are failing. Therefore, non-interpretable models may not constitute a hazard, we consider that it is important to mention its influence at this phase.

As stated, the **Model Development** phase is characterized by a series of decisions considering ML models, techniques, and parameters. In particular, DL technologies used in self-driving vehicles rely on Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN), and Deep Reinforcement Learning (DRL) [71]. CNNs are mainly used for processing spatial information, such as images, and can be viewed as image feature extractors; RNNs are especially good at processing temporal sequence data, such as text, or video streams; and DRL is a combination between reinforcement learning, which uses sequential trial and error to learn the best action to take in every situation, and deep learning. As mentioned in Section 3.4, these models are integrated in perception and localization components. For those tasks, **models** and **(hyper)parameters** were already broadly explored in the literature, considering their computational power and **bias**. A complete review can be found in the work of Grigorescu et al. [71]. Due to the advance in the literature regarding these approaches, we consider that for the application scenario that is being considered, the safety hazards such as **model mismatch**, **(hyper)parameter mismatch**, and **bias** are non safety-critical.

Regarding **error rate**, it is known that a model periodically fails. Due to differences between the ML model runtime and the unpredictability of human behavior, a scenario where an incorrect prediction was made could not compromise a safe operation. For example, considering that a person is in front of the vehicle, but the system does not detect the obstacle/person (i.e., makes an incorrect prediction). The runtime of the system keeps the vehicle moving forward; however, the unpredictability of the human behavior makes the person move out of the front of the vehicle just in time for no accident to occur. The system failed "silently" and safety was not compromised. Since no failure was visibly detected, the error will not be reported, hindering the identification of these types of safety-critical scenarios.

Considering **interpretability**, some of the networks, for example, CNN, can be roughly understood since they have very approximate analogies to different parts of the human visual cortex [71]. For example, interpretable visualization for self-driving vehicles can use a visual attention model, highlighting regions that causally influence deep neural perception [72]. On the other hand, DRL and RNN lack interpretability [73,74].

4.4. Model Testing and Verification

The **Model Testing and Verification** phase is responsible for evaluating the trained model on a dataset that was never seen during the training phase, in order to see how the model generalizes to new input data. To design reliable systems, engineers engage in both testing and verification. At the end of this phase, a verified model and a verification result should be provided to the next phase containing sufficient information to allow potential users to decide if the model is suitable for the intended application. The importance of the **Model Testing and Verification** phase is enhanced if its results provide information that supports the correction of errors (e.g., a testing or verification requirement is not fulfilled and points to a process that needs to be improved, for example, more data needs to be collected).

4.4.1. Activities

Here, we refer to the term testing, meaning the evaluation of the system under various conditions and observing its behavior while watching for defects. The term *verification* relies on producing a compelling argument that the system will not misbehave under a broad range of circumstances. This way, this phase balances two different activities: model testing and formal verification.

The model testing activity is characterized by the systematic evaluation of the model generality and quality, when the model completes training with its decision logic determined. It is important to mention that the testing activity in the AI community mainly considers whether the obtained ML model generalizes to the prepared test dataset, to obtain high test accuracy (or another metric selected to be optimized)—mentioned by some authors as test-based verification [28]. Therefore, this activity involves providing test cases to the trained model and checking the outputs against the expected results [16].

On the other hand, the formal verification activity considers a more general evaluation scope, such as generality, robustness, defect detection, as well as other non-functional requirements (e.g., efficiency) [28], using mathematical techniques to provide evidence that the model satisfies the formal specified properties [75]. Formal verification methods such as model checking and mathematical proof, enable these properties to be rigorously established before the ML model is considered suitable for integration into a safety-critical system.

4.4.2. Hazards Identification

ML testing and verification introduce challenges that arise from the fundamentally different nature and construction of its systems compared to traditional (relatively more deterministic and less statistically-orientated) software systems [75]. At this phase, hazards can be identified regarding:

- **Incompleteness**. The definition of testing/verification requirements can be inadequate and/or not sufficient. As mentioned before, this step is performed during the **Requirements** but it impacts the **Model Testing and Verification** phase. Other important aspects related to incompleteness is the case of limited test scenarios. Due to the large input space, it is difficult to test or approximate all possible inputs (the unknown is never tested). This way, the ML model only encounters a finite number of test samples and the actual operational risk is an empirical quantity of the test set. Thus, the operational risk may be much larger than the identifiable actual risk for the test set, not being representative of the real-world operation performance [21].
- **Non-representative distribution**. Keeping the goal of achieving a certain performance, the test set should also be adequately distributed, in order to cover a balanced number of all the possible scenarios (the ones we can think of). This hazard distinguishes itself from the previous, since it assumes that the amount of test scenarios is enough, and it is mainly concerned with its distribution (the need to make accurate predictions in a representative/well distributed set).

Considering our example application, the **incompleteness** on the limited test scenarios problem is also present because it is demanding to test for every possible scenario of a self-driving vehicle in a factory warehouse. As mentioned before, most unsafe scenarios have low probability and are rare and unpredictable. In the factory context, situations that were never watched in the training phase, such as having more people than usual, having new equipment, or placing objects into different locations, could result in a wrong prediction or in a failure of one of the ML components of the self-driving vehicle.

The **non-representative distribution** of these examples hinders to assure that the model achieves a reliable performance. For example, if we want to have a self-driving vehicle on the shop floor that always achieves a minimum level of performance, we need the test set and, consequently, the testing, to include a balanced distribution of all possible scenarios (even corner and edge cases).

As expected, the examples presented in this phase with respect to the **incompleteness** and the **distribution** hazards are very similar to the **distribution** and **size** hazards detailed in the **Data Management** phase. To a certain extent, we can say that these hazards are exclusively introduced during data collection activity, and its effects spread along the ML lifecycle. Therefore, it is always important to have enough and distributed data for both model training and testing. The scenarios of unknown/unexpected, corner or edge cases, and/or adversarial attacks, will be consistently vulnerable during the entire ML lifecycle due to its data-driven nature.

Assuring that an ML-based safety-critical system will perform adequately in its intended operational environment is a critical part of overall system design. According to [26], ensuring that training and testing datasets are complete require at least ensuring that all aspects of the operational domain have been addressed either by ensuring safe system operation or by ensuring that the system can recognize and mitigate an excursion beyond the defined operational domain. Considering the autonomous vehicles use-case, works such as the PEGASUS project [76] have proposed databases that aim at capturing the challenges imposed by the world; however, ref. [26] suggests to identify a much richer set of relevant safety concerns, focusing on operational domain characterization in terms of objects and events. Furthermore, Safety Of The Intended Functionality (SOTIF) and shorthand for ISO/PAS 21448) [77] provided test cases with high coverage of scenarios and detailed guidelines for data collection, as well as specified conditions that are normally rare and less represented in normal driving but that might impact perception [78].

In a general way, the problem of having a sufficient dataset size in absolute terms is almost impractical to define, as the generality of range, size and complexity of applications is vast [79]. Nevertheless, the effectiveness of a large dataset alone is no guarantee of functional safety, completeness is also related to ensuring that the data used in training has possibly covered all important scenarios [27]. Understating completeness across the operation domain remains an open challenge [16].

4.5. Model Deployment

The **Model Deployment** phase receives as input a verified model that is ready to be deployed into a target platform. The outcome of this phase is a fully deployed and operating real-world system.

4.5.1. Activities

The first step of **Model Deployment** is integration of the ML model into a wider system architecture. Here, the integration must consider the architecture of the system into which the model is being deployed (which includes linking the model to system inputs, e.g., sensor systems, as well as protecting the wider system from hazardous effects that may arise from incorrect model outputs).

After integration, the monitoring of the deployed model ensures that it continues to perform as intended. For that purpose, the monitoring activity considers four types of monitoring: (1) input monitoring, for checking whether inputs are within acceptable bounds before they are given to the ML model; (2) environment monitoring, for checking that the observed environment matches any assumptions made during the ML workflow; (3) model internal monitoring, to protect against the effects of single event upsets; (4) output monitoring, by replicating a traditional system safety approach in which a high-integrity monitor is used alongside a lower-integrity item [16].

Lastly, since new data are being collected continuously, ML models can become obsolete when they are not maintained properly. This concept is called model drift. In order to avoid this, and similarly to software, it is expected that the deployed ML models need to be updated during a system life; therefore, updating is also an existing activity (offline maintenance). Moreover, this activity can also include, as a particular case, updates that occur as part of online learning. Online learning models can learn from new examples in close to real time. That is, they predict and train in real time (typically one data point at a time is considered for updating the parameters and re-training the model, which is immediately implemented) [36].

4.5.2. Hazards Identification

When deploying the ML model into different platforms or with different implementation frameworks, there will always be opportunities for introducing hazards:

- **Differences in computation platforms**. Deploying a model into a device can result in computation limitations and compatibility issues across platforms. For example, for a neural network, which is mostly developed and trained on servers or computers with Graphic Processing Unit (GPU) support,

when it needs to be deployed on a mobile device or edge device with limited computation power, the software must be adjusted for computation/energy efficiency, which could lead to computation differences affecting system behavior (e.g., time performance).

- **Operational environment.** Differences between the operational environment and data used for model development and testing, can lead to different/new inputs that affect the output produced. This could happen due to several reasons: (1) failure on one of the subsystems that provide inputs to the deployed ML model, (2) deliberate actions of an adversary; and (3) changes in the underlying processes to which the data are related (changes on the environment or on the way people or other systems behave) [16].
- **Non detection of potentially incorrect outputs**. An ML model may produce an incorrect output when it is used outside the intended operational domain, which could be detected during monitoring. For that purpose, ML-based systems calculate a "measure of confidence" (mentioned in the **Requirements** phase as the **inadequate safe operating values** hazard). If this value is incorrectly defined and/or implemented, its consequences are felt during the **Model Deployment** phase, where there is a non-safe confidence in the model and potentially incorrect outputs may not be detected.
- **New data/Continuous learning**. This hazard only considers the case of online learning (i.e., systems that continue to learn parameters and train the model during operation). Despite the fact that the incorporation of new data from the real operation domain suggests improving the model performance, since new data are added to the model training, the dataset distribution could be biased and it is no longer supervised, susceptible to result in lower model performance in scenarios that are no longer as frequent on the new data (e.g., a self-driving vehicle that was trained before operation on an adequate distributed dataset is now operating only at dark scenarios; for this case, the model could start to be optimized for dark conditions and to behave less accurate in the remaining day time scenarios).

Regarding the **Model Deployment** phase, the **differences in computation platforms** where ML models are deployed could be crucial in terms of safety if it causes differences in the runtime. For example, if some unexpected obstacle appears in front of the self-driving vehicle, it is important that the vehicle has the capacity to provide a prediction within an acceptable amount of time; otherwise, the behavioral planning might fail and not be able to change its route on time to avoid the obstacle.

Furthermore, once deployed and running on the self-driving vehicle, the ML model could face some unexpected changes, and have to deal with different **operational environments**. This can happen due to several reasons: there is a failure on one of the subsystem modules compromising data acquisition (i.e., data are corrupted or do not have the expected quality); an adversary attack happens that was not accounted for during data acquisition (already mentioned in the **Data Management** phase); or there is a change in the environment or the behavior of a human worker changes for some reason, where none of the situations were accounted for before (e.g., walls and/or floor of the factory are re-painted with a different color, or workers need to change their behavior due to new functions that they need to start to perform). All of these examples of differences between the operational use and the data acquired for the model development and testing/verification, could lead to incorrect predictions and affect the safety of the robot, human, and environment.

On an additional topic, online learning has been pointed out as a practical and useful tool for solving large-scale decision-making problems in many systems and networking domains, including spam filtering, network intrusion detection, E-commerce, and the social networking industry [80]. Nevertheless, online learning methods are difficult to evaluate online since it is not possible to hold out a "test" set for evaluation because no distributional assumptions are being made. As pointed out above, the inability to control the distribution makes this type of algorithm currently not suitable for safety-critical applications as the ones we are considering in this paper. However, this may change as research also evolves in this area, and measures become available that explicitly address the related hazards.

5. Conclusions

The application of ML in safety-critical systems in domains such as healthcare, transportation, and manufacturing, has great potential. However, several concerns regarding standards and norms to ensure their safe operation are still open topics among the research community.

Control systems for safety-critical applications must comply with the approval conditions of their area of application. Therefore, most industry branches have their own standards and norms. Some progress has already been made for the definition of safety goals in safety-critical applications, providing evidence that ML is sufficiently safe for its intended use. Specifically, work developed for the automotive industry covers a significant fraction of safety-related concerns. Despite the fact that this work ranges from high-level suggestions governing behavior at the system level (e.g., vehicle level) to low-level approaches that attempt to find a solution at the ML component, it does not yet adequately address every individual hazard identified in our paper. This indicates that further work is required to cover all of the safety-related aspects of the ML lifecycle, and to support the use of ML-based control systems in highly safety-critical applications and their certification.

Our paper focuses on presenting the challenges during the design and development of a system with an integrated ML algorithm, specifying for each phase of the ML lifecycle which safety hazard can be introduced during the specific activities of that phase. By providing an extensive hazard analysis, and by illustrating each hazard with a real-world application example, our work aims to establish the basis for a future definition of product-oriented (i.e., describing technical requirements for approval) and process-oriented (i.e., defining processes to be observed) precautions in order to reduce hazardous behavior and its consequences. This rigorous approach also provides a natural basis for the creation of safety cases that demonstrate the safety of ML-based CPS by supplying sufficient evidence against each of the hazards identified herein.

We believe that our work is a step forward into future holistic approaches for safety engineering and certification of ML-based systems, supporting the research community to identify remaining gaps regarding product- or process-related measures that are required to ensure the proper functioning of ML-based CPS in safety-critical applications.

Author Contributions: Conceptualization, C.T.; investigation, A.P.; methodology, A.P. and C.T.; supervision, C.T.; writing—original draft, A.P.; writing—review and editing, A.P. and C.T. All authors have read and agreed to the published version of the manuscript.

Funding: This paper has been created within the framework of the ITEA project 17032 CyberFactory#1. The German partners are funded by the German Federal Ministry of Education and Research (Grant Number 01IS18061D).

Acknowledgments: The figures present in this paper were designed using resources from Flaticon.com (http://www.flaticon.com).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dreossi, T.; Donzé, A.; Seshia, S.A. Compositional falsification of cyber-physical systems with machine learning components. *J. Autom. Reason.* **2019**, *63*, 1031–1053. [CrossRef]
2. Shi, J.; Wan, J.; Yan, H.; Suo, H. A survey of cyber-physical systems. In Proceedings of the 2011 International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, China, 9–11 November 2011; pp. 1–6.
3. Lu, Y. Cyber physical system (CPS)-based industry 4.0: A survey. *J. Ind. Integr. Manag.* **2017**, *2*, 1750014. [CrossRef]
4. Zheng, P.; Sang, Z.; Zhong, R.Y.; Liu, Y.; Liu, C.; Mubarok, K.; Yu, S.; Xu, X.; Wang, H. Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives. *Front. Mech. Eng.* **2018**, *13*, 137–150. [CrossRef]
5. Thoben, K.D.; Wiesner, S.; Wuest, T. "Industrie 4.0" and smart manufacturing-a review of research issues and application examples. *Int. J. Autom. Technol.* **2017**, *11*, 4–16. [CrossRef]

6. Heng, S. Industry 4.0: Upgrading of Germany's Industrial Capabilities on the Horizon. 2014. Available online: https://ssrn.com/abstract=2656608 (accessed on 14 November 2020).
7. Robla-Gómez, S.; Becerra, V.M.; Llata, J.R.; Gonzalez-Sarabia, E.; Torre-Ferrero, C.; Perez-Oria, J. Working together: A review on safe human-robot collaboration in industrial environments. *IEEE Access* **2017**, *5*, 26754–26773. [CrossRef]
8. Oyekan, J.O.; Hutabarat, W.; Tiwari, A.; Grech, R.; Aung, M.H.; Mariani, M.P.; López-Dávalos, L.; Ricaud, T.; Singh, S.; Dupuis, C. The effectiveness of virtual environments in developing collaborative strategies between industrial robots and humans. *Robot. Comput. Integr. Manuf.* **2019**, *55*, 41–54. [CrossRef]
9. Evjemo, L.D.; Gjerstad, T.; Grøtli, E.I.; Sziebig, G. Trends in Smart Manufacturing: Role of Humans and Industrial Robots in Smart Factories. *Curr. Robot. Rep.* **2020**, *1*, 35–41. [CrossRef]
10. Gharib, M.; Lollini, P.; Botta, M.; Amparore, E.; Donatelli, S.; Bondavalli, A. On the Safety of Automotive Systems Incorporating Machine Learning Based Components: A Position Paper. In Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Luxembourg, 25–28 June 2018; pp. 271–274.
11. Faria, J.M. Machine learning safety: An overview. In Proceedings of the 26th Safety-Critical Systems Symposium, York, UK, 6–8 February 2018.
12. Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; Mané, D. Concrete problems in AI safety. *arXiv* **2016**, arXiv:1606.06565.
13. Serban, A.C. Designing Safety Critical Software Systems to Manage Inherent Uncertainty. In Proceedings of the 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), Hamburg, Germany, 25–29 March 2019; pp. 246–249.
14. Hains, G.J.; Jakobsson, A.; Khmelevsky, Y. Formal methods and software engineering for DL. Security, safety and productivity for DL systems development. *arXiv* **2019**, arXiv:1901.11334.
15. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: New York, NY, USA, 2006.
16. Ashmore, R.; Calinescu, R.; Paterson, C. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *arXiv* **2019**, arXiv:1905.04223.
17. Kumeno, F. Sofware engneering challenges for machine learning applications: A literature review. *Intell. Decis. Technol.* **2020**, *13*, 463–476. [CrossRef]
18. Jenn, E.; Albore, A.; Mamalet, F.; Flandin, G.; Gabreau, C.; Delseny, H.; Gauffriau, A.; Bonnin, H.; Alecu, L.; Pirard, J.; et al. Identifying Challenges to the Certification of Machine Learning for Safety Critical Systems. In Proceedings of the 10th European Congress on Embedded Real Time Systems (ERTS), Toulouse, France, 29–31 January 2020.
19. G-34, Artificial Intelligence in Aviation. Available online: https://www.sae.org/works/committeeHome.do?comtID=TEAG34 (accessed on 22 October 2020).
20. Varshney, K.R. Engineering safety in machine learning. In Proceedings of the 2016 Information Theory and Applications Workshop (ITA), La Jolla, CA, USA, 31 January–5 February 2016; pp. 1–5.
21. Varshney, K.R.; Alemzadeh, H. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big Data* **2017**, *5*, 246–255. [CrossRef] [PubMed]
22. Kläs, M.; Vollmer, A.M. Uncertainty in machine learning applications: A practice-driven classification of uncertainty. In *International Conference on Computer Safety, Reliability, and Security*; Springer: Västerås, Sweden, 2018; pp. 431–438.
23. Salay, R.; Queiroz, R.; Czarnecki, K. An analysis of ISO 26262: Using machine learning safely in automotive software. *arXiv* **2017**, arXiv:1709.02435.
24. Borg, M.; Englund, C.; Wnuk, K.; Duran, B.; Levandowski, C.; Gao, S.; Tan, Y.; Kaijser, H.; Lönn, H.; Törnqvist, J. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. *arXiv* **2018**, arXiv:1812.05389.
25. Henriksson, J.; Borg, M.; Englund, C. Automotive Safety and Machine Learning: Initial Results from a Study on How to Adapt the ISO 26262 Safety Standard. In Proceedings of the 2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS), Gothenburg, Sweden, 28 May 2018; pp. 47–49.
26. Koopman, P.; Fratrik, F. How many operational design domains, objects, and events? In Proceedings of the SafeAI@ AAAI, Honolulu, HI, USA, 27 January 2019.

27. Schwalbe, G.; Schels, M. A Survey on Methods for the Safety Assurance of Machine Learning Based Systems. In Proceedings of the 10th European Congress on Embedded Real Time Software and Systems (ERTS 2020), Toulouse, France, 29–31 January 2020.
28. Ma, L.; Juefei-Xu, F.; Xue, M.; Hu, Q.; Chen, S.; Li, B.; Liu, Y.; Zhao, J.; Yin, J.; See, S. Secure deep learning engineering: A software quality assurance perspective. *arXiv* **2018**, arXiv:1810.04538.
29. Aravantinos, V.; Diehl, F. Traceability of deep neural networks. *arXiv* **2018**, arXiv:1812.06744.
30. Avizienis, A.; Laprie, J.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.* **2004**, *1*, 11–33. [CrossRef]
31. Smith, D.; Simpson, K. *Functional Safety*; Routledge: London, UK, 2004.
32. Möller, N.; Hansson, S.O. Principles of engineering safety: Risk and uncertainty reduction. *Reliab. Eng. Syst. Saf.* **2008**, *93*, 798–805. [CrossRef]
33. Möller, N. The concepts of risk and safety. In *Handbook of Risk Theory: Epistemology, Decision Theory, Ethics, and Social Implications of Risk*; Springer: Dordrecht, The Netherlands, 2012; Volume 1.
34. Leveson, N.G. *Engineering a Safer World: Systems Thinking Applied to Safety*; The MIT Press: Cambridge, MA, USA, 2016.
35. Kelly, T. Arguing Safety—A Systematic Approach to Safety Case Management. Ph.D. Thesis, Department of Computer Science, University of York, York, UK, 1998.
36. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*; O'Reilly Media: Newton, MA, USA, 2019.
37. Witten, I.H.; Frank, E. Data mining: Practical machine learning tools and techniques with Java implementations. *ACM Sigmod Rec.* **2002**, *31*, 76–77. [CrossRef]
38. Dey, A. Machine learning algorithms: A review. *Int. J. Comput. Sci. Inf. Technol.* **2016**, *7*, 1174–1179.
39. James, G.; Witten, D.; Hastie, T.; Tibshirani, R. *An Introduction to Statistical Learning*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 112, ISBN 978-1-4614-7138-7.
40. Patterson, J.; Gibson, A. *Deep Learning: A Practitioner's Approach*; O'Reilly Media, Inc.: Newton, MA, USA, 2017.
41. International Standards Organization. *ISO 26262:2018-12: Road Vehicles—Functional Safety*; International Standards Organization: Geneva, Switzerland, 2018.
42. The International Electrotechnical Commission. *IEC 61508:2010: Functional Safety of Electrical/Electronic/ Programmable Electronic Safety-Related Systems*; The International Electrotechnical Commission: Geneva, Switzerland, 2010.
43. SAE International. *ARP4754A:2010: Guidelines for Development of Civil Aircraft and Systems*; SAE International: Warrendale, PA, USA, 2010.
44. Küpper, D.; Lorenz, M.; Kuhlmann, K.; Bouffault, O.; Heng, L.Y.; Van Wyck, J.; Köcher, S.; Schlagete, J. *AI in the Factory of the Future. The Ghost in the Machine*; The Boston Consulting Group: Boston, MA, USA, 2018.
45. Ansari, F.; Erol, S.; Sihn, W. Rethinking Human-Machine Learning in Industry 4.0: How Does the Paradigm Shift Treat the Role of Human Learning? *Procedia Manuf.* **2018**, *23*, 117–122. [CrossRef]
46. Kato, S.; Takeuchi, E.; Ishiguro, Y.; Ninomiya, Y.; Takeda, K.; Hamada, T. An Open Approach to Autonomous Vehicles. *IEEE Micro* **2015**, *35*, 60–68. [CrossRef]
47. Zaharia, M.; Chen, A.; Davidson, A.; Ghodsi, A.; Hong, S.; Konwinski, A.; Murching, S.; Nykodym, T.; Ogilvie, P.; Parkhe, M.; et al. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* **2018**, *41*, 39–45.
48. Vogelsang, A.; Borg, M. Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), Jeju Island, Korea, 23–27 September 2019; pp. 245–251.
49. Ishikawa, F.; Yoshioka, N. How Do Engineers Perceive Difficulties in Engineering of Machine-Learning Systems?—Questionnaire Survey. In Proceedings of the 2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER IP), Montreal, QC, Canada, 28 May 2019; pp. 2–9.
50. Davis, A.M. *Software Requirements: Objects, Functions, and States*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1993.
51. Ellims, M.; Barbier, E.; Botham, J. Safety Analysis Process for Machine Learning in Automated Vehicle Software. In Proceedings of the 26th ITS World Congress, Singapore, 21–25 October 2019.

52. Gharib, M.; Bondavalli, A. On the Evaluation Measures for Machine Learning Algorithms for Safety-Critical Systems. In Proceedings of the 2019 15th European Dependable Computing Conference (EDCC), Naples, Italy, 17–20 September 2019; pp. 141–144.
53. Reschka, A. Safety concept for autonomous vehicles. In *Autonomous Driving*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 473–496.
54. Roh, Y.; Heo, G.; Whang, S.E. A Survey on Data Collection for Machine Learning: A Big Data—AI Integration Perspective. *IEEE Trans. Knowl. Data Eng.* **2019**, doi:10.1109/TKDE.2019.2946162. [CrossRef]
55. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
56. Chen, X.; Fang, H.; Lin, T.Y.; Vedantam, R.; Gupta, S.; Dollár, P.; Zitnick, C.L. Microsoft coco captions: Data collection and evaluation server. *arXiv* **2015**, arXiv:1504.00325.
57. Patki, N.; Wedge, R.; Veeramachaneni, K. The Synthetic Data Vault. In Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Montreal, QC, Canada, 17–19 October 2016; pp. 399–410.
58. Zhang, S.; Zhang, C.; Yang, Q. Data preparation for data mining. *Appl. Artif. Intell.* **2003**, *17*, 375–381. [CrossRef]
59. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. *Microsoft Coco: Common Objects in Context*; European Conference on Computer Vision; Springer: Charm, Switzerland, 2014; pp. 740–755.
60. Japkowicz, N.; Stephen, S. The class imbalance problem: A systematic study. *Intell. Data Anal.* **2002**, *6*, 429–449. [CrossRef]
61. Koopman, P.; Wagner, M. Autonomous Vehicle Safety: An Interdisciplinary Challenge. *IEEE Intell. Transp. Syst. Mag.* **2017**, *9*, 90–96. [CrossRef]
62. Goodfellow, I.; McDaniel, P.; Papernot, N. *Making Machine Learning Robust against Adversarial Inputs*; Communications of the ACM: New York, NY, USA, 2018; pp. 56–66.
63. Margaret, M. Bias in the Vision and Language of Artificial Intelligence. Available online: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture19-bias.pdf (accesed on 18 November 2020).
64. Gu, X.; Easwaran, A. Towards Safe Machine Learning for CPS: Infer Uncertainty from Training Data. In Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'19), Montreal, QC, Canada, 16–18 April 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 249–258. [CrossRef]
65. Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
66. Roscher, R.; Bohn, B.; Duarte, M.F.; Garcke, J. Explainable machine learning for scientific insights and discoveries. *IEEE Access* **2020**, *8*, 42200–42216. [CrossRef]
67. Miao, H.; Li, A.; Davis, L.S.; Deshpande, A. Towards Unified Data and Lifecycle Management for Deep Learning. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 571–582.
68. How to Select Algorithms for Azure Machine Learning. Available online: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-select-algorithms (accessed on 24 August 2020).
69. Molnar, C. *Interpretable Machine Learning*; Lulu.com: Morrisville, NC, USA, 2020.
70. Marcus, G. Deep learning: A critical appraisal. *arXiv* **2018**, arXiv:1801.00631.
71. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **2020**, *37*, 362–386. [CrossRef]
72. Kim, J.; Canny, J. Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 December 2017.
73. Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; et al. Relational deep reinforcement learning. *arXiv* **2018**, arXiv:1806.01830.
74. Brown, A.; Tuor, A.; Hutchinson, B.; Nichols, N. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In Proceedings of the First Workshop on Machine Learning for Computing Systems, Tempe, AZ, USA, 12 June 2018; pp. 1–8.
75. Zhang, J.M.; Harman, M.; Ma, L.; Liu, Y. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Trans. Softw. Eng.* **2020**. [CrossRef]

76. Pütz, A.; Zlocki, A.; Bock, J.; Eckstein, L. System validation of highly automated vehicles with a database of relevant traffic scenarios. *Situations* **2017**, *1*, 19–22.
77. ISO. *PAS 21448-Road Vehicles-Safety of the Intended Functionality*; International Organization for Standardization: Geneva, Switzerland, 2019.
78. Mariani, R. Challenges in AI/ML for Safety Critical Systems. Available online: http://www.dfts.org/_2019/DFT_2019-Mariani-v3.pdf (accessed on 14 November 2020).
79. Sutherland, G.; Hessami, A. Safety Critical Integrity Assurance in Large Datasets. In Proceedings of the 28th Safety-Critical Systems Symposium, York, UK, 11–13 February 2020; p. 308.
80. Huang, L.; Joseph, A.D.; Nelson, B.; Rubinstein, B.I.; Tygar, J.D. Adversarial Machine Learning. In Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISec '11), Chicago, IL, USA, 17–21 October 2011; pp. 43–58. [CrossRef]