Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges

ROB ASHMORE†, Defence Science and Technology Laboratory, UK
RADU CALINESCU and COLIN PATERSON, University of York and Assuring Autonomy International Programme, UK

Machine learning has evolved into an enabling technology for a wide range of highly successful applications. The potential for this success to continue and accelerate has placed machine learning (ML) at the top of research, economic, and political agendas. Such unprecedented interest is fuelled by a vision of ML applicability extending to healthcare, transportation, defence, and other domains of great societal importance. Achieving this vision requires the use of ML in safety-critical applications that demand levels of assurance beyond those needed for current ML applications. Our article provides a comprehensive survey of the state of the art in the *assurance of ML*, i.e., in the generation of evidence that ML is sufficiently safe for its intended use. The survey covers the methods capable of providing such evidence at different stages of the *machine learning lifecycle*, i.e., of the complex, iterative process that starts with the collection of the data used to train an ML component for a system, and ends with the deployment of that component within the system. The article begins with a systematic presentation of the ML lifecycle and its stages. We then define assurance desiderata for each stage, review existing methods that contribute to achieving these desiderata, and identify open challenges that require further research.

CCS Concepts: • **Computing methodologies → Machine learning**; **Model verification and validation**; • **General and reference → Surveys and overviews**;

Additional Key Words and Phrases: Machine learning lifecycle, machine learning workflow, safety-critical systems, assurance, assurance evidence

ACM Reference format:
Rob Ashmore, Radu Calinescu, and Colin Paterson. 2021. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *ACM Comput. Surv.* 54, 5, Article 111 (May 2021), 39 pages.
https://doi.org/10.1145/3453444

1 INTRODUCTION

The recent success of **machine learning (ML)** has taken the world by storm. While far from delivering the humanlike intelligence postulated by Artificial Intelligence pioneers [39], ML techniques

Rob Ashmore, Radu Calinescu, and Colin Paterson are contributed equally to the article.

†Research carried out while the author was a Visiting Fellow of the Assuring Autonomy International Programme.

This work was partly funded by the Assuring Autonomy International Programme and the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'.

Authors' addresses: R. Ashmore, East Court - C Grenville Building, DSTL, Portsdown West, Hill Road, Fareham, Hants, PO17 6AD, England; email: rdashmore@dstl.gov.uk; R. Calinescu and, C. Paterson, Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, England; emails: {radu.calinescu, colin.paterson}@york.ac.uk.

0360-0300/2021/05-ART111 $15.00
https://doi.org/10.1145/3453444

such as deep learning have remarkable applications. The use of these techniques in products ranging from smart phones [8, 137] and household appliances [79] to recommender systems [36, 139] and automated translation services [186] has become commonplace. There is a widespread belief that this is just the beginning of an ML-enabled technological revolution [55, 107]. Stakeholders as diverse as researchers, industrialists, policy makers and the general public envisage that ML will soon be at the core of novel applications and services used in healthcare, transportation, defence and other key areas of economy and society [11, 41, 74, 90, 111].

Achieving this vision requires a step change in the level of assurance provided for ML. The occasional out-of-focus photo taken by an ML-enabled smart camera can easily be deleted, and the selection of an odd word by an automated translation service is barely noticed. Although increasingly rare, similar ML errors may be unacceptable in medical diagnosis applications or self-driving cars. For such safety-critical systems, ML errors can lead to failures that cannot be reverted or ignored, and ultimately cause harm to their users or operators. Therefore, the use of ML to synthesise components of safety-critical systems must be assured by evidence that these *ML components* are fit for purpose *and* adequately integrated into their systems. This evidence must be sufficiently thorough. Where required by the domain, it must enable the creation of compelling *assurance cases* [21, 122] that explain why the systems can be trusted for their intended applications. Given this need, within this article, we use the term *assurance* to refer to evidence that supports safety arguments, rather than in a more general quality-related manner. Although we focus on evidence, we note that the argument structure needs to be credible [92].

Our article represents the first survey of the methods available for obtaining evidence concerning ML components for use in assurance cases, with a focus on ML techniques that synthesise system components (e.g., classifiers) from incomplete training data. As with any engineering artefact, assurance can only be provided by understanding the complex, iterative process employed to produce and use ML components, i.e., the *machine learning lifecycle*. We therefore start by defining this lifecycle, which consists of four interrelated stages carried out following a spiral process model [22]. A number of activities are associated with each stage. The ML lifecycle stages and activities presented in the article are derived from our study of the existing research literature, and our practical experience from over 15 Assuring Autonomy International Programme projects on the assurance and regulation of robotics and autonomous systems.[1]

The first stage, **Data Management (DM)**, focuses on obtaining the datasets required for the training and for the verification of the ML components. This stage includes activities ranging from data collection to data preprocessing [94] (e.g., labelling) and augmentation [140]. The second stage, *Model Learning*, comprises the activities associated with synthesis of the ML component starting from the training dataset. The actual machine learning happens in this stage, which also includes activities such as selection of the ML algorithm and hyperparameters [17, 166]. The third stage, **Model Verification (MV)**, is responsible for providing evidence to demonstrate that the synthesised ML component complies with its requirements. This stage is essential for the ML components of safety-critical systems. Finally, the last stage of the ML lifecycle is *Model Deployment*. This stage focuses on the integration and operation of the ML component within a fully fledged system.

To ensure a systematic coverage of ML assurance methods, we structure our survey based on the assurance considerations that apply at the four stages of the ML lifecycle. For each stage, we identify the assurance-related *desiderata* (i.e., the key assurance requirements, derived from the body of research covered in our survey) for the artefacts produced by that stage. The relative importance of these desiderata will be largely context dependent. Indeed, it may be necessary for the

[1] https://www.york.ac.uk/assuring-autonomy/projects/.

engineer to agree where it is appropriate to find tradeoffs between desiderata within, and between, ML lifecycle stages. By considering all of the proposed desiderata, with respect to the operating context, a more compelling assurance argument may be constructed. We then present the methods available for achieving these desiderata, with their assumptions, advantages and limitations. This represents an analysis of over two decades of sustained research on ML methods for data management, model learning, verification, and deployment. Finally, we determine the open challenges that must be addressed through further research to fully satisfy the stage desiderata and to enable the use of ML components in safety-critical systems. Much like the desiderata, the relative importance of these challenges will be dependent on the operating context, and hence ranking them is outside of scope of this work.

Our survey and the machine learning lifecycle underpinning its organisation are relevant to a broad range of ML types, including supervised, unsupervised and reinforcement learning. Necessarily, some of the methods presented in the survey are only applicable to specific types of ML; we clearly indicate where this is the case. As such, the survey supports a broad range of ML stakeholders, ranging from practitioners developing convolutional neural networks for the classification of road signs in self-driving cars, to researchers devising new ensemble learning techniques for safety-critical applications, and to regulators managing the introduction of systems that use ML components into everyday use.

The rest of the article is structured as follows. In Section 2, we present the machine learning lifecycle, describing the activities encountered within each of its stages and introducing ML terminology used throughout the article. In Section 3, we overview the few existing surveys that discuss verification, safety or assurance aspects of machine learning. We explain that each of these surveys focuses on a specific ML lifecycle stage or subset of activities, and/or addresses only a narrow aspect of ML assurance. The ML assurance desiderata, methods and open challenges for the four stages of the ML lifecycle are then detailed in Sections 4 to 7. Together, these sections provide a comprehensive set of guidelines for the developers of safety-critical systems with ML components, and inform researchers about areas where additional ML assurance methods are needed. We conclude the article with a brief summary in Section 8.

2 THE MACHINE LEARNING LIFECYCLE

Machine learning provides mechanisms for the extraction of *models* (or *patterns*) from data [20, 60, 120]. In this article, we are concerned with the use of such ML models in safety-critical systems, e.g., to enable these systems to understand the environment they operate in, and to decide their response to changes in the environment. Assuring this use of ML models requires an in-depth understanding of the *machine learning lifecycle*, i.e., of the process used for their development and integration into a fully fledged system. Like traditional system development, this process is underpinned by a set of system-level requirements, from which the requirements and operating constraints for the ML models are derived. As an example, the requirements for a ML model for the classification of British road signs can be derived from the high-level requirements for a self-driving car intended to be used in the UK. However, unlike traditional development processes, the development of ML models involves the acquisition of datasets, and *experimentation* [114, 189], i.e., the manipulation of these datasets and the use of ML *training* techniques to produce models of the data that optimise error functions derived from requirements. This experimentation yields a processing pipeline capable of taking data as input and of producing ML models that, when integrated into the system and applied to data unseen during training, achieve their requirements in the deployed context.

As shown in Figure 1, the machine learning lifecycle consists of four stages. The first three stages—*Data Management*, *Model Learning*, and *Model Verification*—comprise the activities by

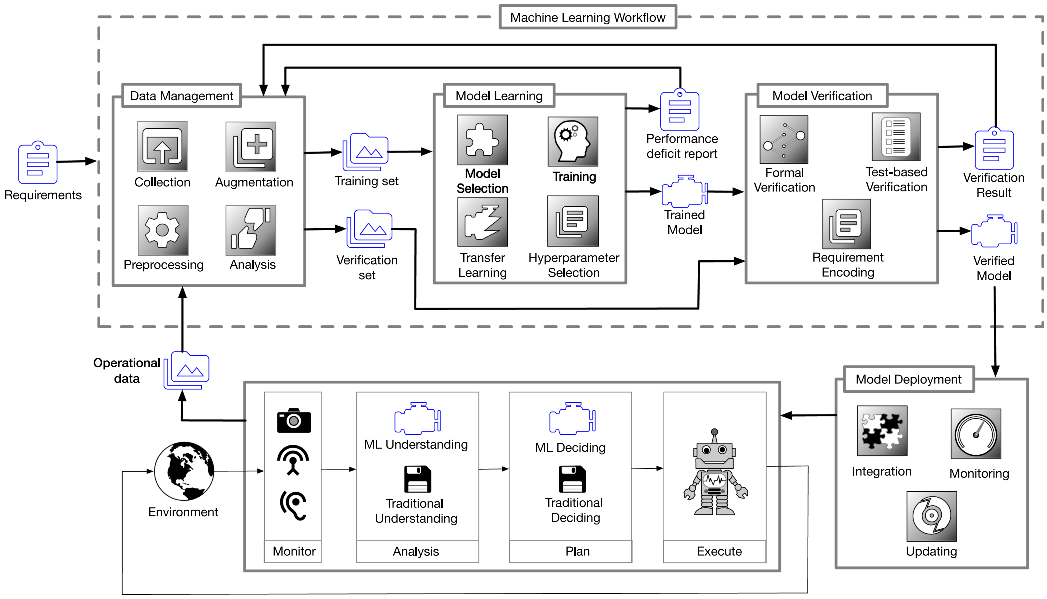

Fig. 1. The machine learning lifecycle comprises four stages performed using a spiral process model.

which machine learned models are produced. Accordingly, we use the term *machine learning workflow* to refer to these stages taken together. The fourth stage, *Model Deployment*, comprises the activities concerned with the deployment of ML models within an operational system, alongside components obtained using traditional software and system engineering methods. We provide brief descriptions of each of these stages below.

Data are at the core of any application of machine learning. As such, the ML lifecycle starts with a Data Management stage. This stage is responsible for the acquisition of the data underpinning the synthesis of machine learned models that can then be used "to predict future data, or to perform other kinds of decision making under uncertainty" [120]. This stage comprises four key activities, and produces the *training dataset* and *verification dataset* used for the training and verification of the ML models in later stages of the ML lifecycle, respectively. The first data management activity, *collection* [59, 174], is concerned with gathering *data samples* through observing and measuring the real-world (or a representation of the real-world) system, process or phenomenon for which an ML model needs to be built. When data samples are unavailable for certain scenarios, or their collection would be too costly, time consuming or dangerous, *augmentation* methods [140, 185] are used to add further data samples to the collected datasets. Additionally, the data collected from multiple sources may be heterogeneous in nature, and therefore *preprocessing* [93, 191] may be required to produce consistent datasets for training and verification purposes. Preprocessing may also seek to reduce the complexity of collected data or to engineer features to aid in training [64, 86]. Furthermore, preprocessing may be required to label the data samples when they are used in supervised ML tasks [59, 60, 120]. The need for additional data collection, augmentation and preprocessing is established through the *analysis* of the data [133].

In the Model Learning stage of the machine learning lifecycle, the ML engineer typically starts by selecting the type of model to be produced. This *model selection* is undertaken with reference to the problem type (e.g., classification or regression), the volume and structure of the training data [113, 152], and often in light of personal experience. A *loss function* is then constructed as a measure of training error. The aim of the *training* activity is to produce an ML model that minimises this error. This requires the development of a suitable data use strategy, so as to determine how much of the training dataset should be held for model validation,[2] and whether all the other data samples should be used together for training or "minibatch methods" that use subsets of data samples over successive training cycles should be employed [60]. The ML engineer is also responsible for *hyperparameter selection*, i.e., for the choosing the parameters of the training algorithm. Hyperparameters control key ML model characteristics such as overfitting, underfitting and model complexity. Finally, when models or partial models that have proved successful within a related context are available, *transfer learning* enables their integration within the new model architecture or their use as a starting point for training [124, 135, 160]. When the resulting ML model achieves satisfactory levels of performance, the next stage of the ML workflow can commence. Otherwise, the process needs to return to the Data Management stage, where additional data are collected, augmented, preprocessed and analysed to improve the training further.

The third stage of the ML lifecycle is Model Verification. The central challenge of this stage is to ensure that the trained model performs well on new, previously unseen inputs (this is known as generalization) [59, 60, 120]. As such, the stage comprises activities that provide evidence of the model's ability to generalise to data not seen during the model learning stage. A *test-based verification* activity assesses the performance of the learned model against the verification dataset that the Data Management stage has produced independently from the training dataset. This dataset will have commonalities with the training data, but it may also include elements that have been deliberately chosen to demonstrate a verification aim, which it would be inappropriate to include in the training data. When the data samples from this set are presented to the model, a *generalization error* is computed [121, 159]. If this error violates performance criteria established by a *requirement encoding* activity, then the process needs to return to either the Data Management stage or the Model Learning stage of the ML lifecycle. Additionally, a *formal verification* activity may be used to verify whether the model complies with a set of formal properties that encode key requirements for the ML component. Formal verification methods such as model checking and mathematical proof allow for these properties to be rigorously established before the ML model is deemed suitable for integration into the safety-critical system. As for failed testing-based verification, further Data Management and/or Model Learning activities are necessary when these properties do not hold. The precise activities required from these earlier stages of the ML workflow are determined by the *verification result*, which summarises the outcome of all verification activities.

Assuming that the verification result contains all the required assurance evidence, a system that uses the now *verified model* is assembled in the Model Deployment stage of the ML lifecycle. This stage comprises activities concerned with the *integration* of verified ML model(s) with system components developed and verified using traditional software engineering methods, with the *monitoring* of its operation, and with its *updating* thorough offline maintenance or online learning. The outcome of the Model Deployment stage is a fully fledged deployed and operating system.

More often than not, the safety-critical systems envisaged to benefit from the use of ML models are autonomous or self-adaptive systems that require ML components to cope with the dynamic and uncertain nature of their operating environments [29, 90, 111]. As such, Figure 1 depicts this type of system as the outcome of the Model Deployment stage. Moreover, the diagram shows two key roles that ML models may play within the established **monitor-analyse-plan-execute**

[2] Model validation represents the frequent evaluation of the ML model during training, and is carried out by the development team to calibrate the training algorithm. This differs essentially from what validation means in software engineering (i.e., an independent assessment performed to establish whether a system satisfies the needs of its intended users).

(MAPE) control loop [72, 84, 110] of these systems. We end this section with a brief description of the MAPE control loop and of these typical uses of ML models within it.

In its four steps, the MAPE control loop senses the current state of the environment through *monitoring*, derives an understanding of the world through the *analysis* of the sensed data, decides suitable actions through *planning*, and then acts upon these plans through *executing* their actions. Undertaking these actions alters the state of the system and the environment in which it operates.

The monitoring step employs hardware and software components that gather data as a set of samples from the environment during operation. The choice of sensors requires an understanding of the system requirements, the intended operational conditions, and the platform into which they will be deployed. Data gathered from the environment will typically be partial and imperfect due to physical, timing and financial constraints.

The analysis step extracts features from data samples as encoded domain-specific knowledge. This can be achieved through the use of ML models combined with traditional software components. The features extracted may be numerical (e.g., blood sugar level in a healthcare system), ordinal (e.g., position in queue in a traffic management system) or categorical (e.g., an element from the set $\{\text{car}, \text{bike}, \text{bus}\}$ in a self-driving car). The features extracted through analysing the datasets obtained during monitoring underpin the understanding of the current state of the environment and that of the system itself.

The planning (or decision) step can employ a combination of ML models and traditional reasoning engines to select a course of action to be undertaken. The action(s) selected will aim to fulfil the system requirements subject to any defined constraints. The action set available is dictated by the capabilities of the system, and is restricted by operating conditions and constraints defined in the requirements specification.

Finally, in the execution step, the system enacts the selected actions through software and hardware effectors and, in doing so, changes the environment within which it is operating. The dynamic and temporal nature of the system and the environment requires the MAPE control loop to be invoked continuously until a set of system-level objectives has been achieved or a stopping criterion was reached.

New data samples gathered during operation can be exploited by the Data Management activities and, where appropriate, new models may be learned and deployed within the system. This deployment of new ML models can be carried out either as an offline maintenance activity, or through the online updating of the operating system.

3 RELATED SURVEYS

The large and rapidly growing body of ML research is summarised by a plethora of surveys. The vast majority of these surveys narrowly focus on a particular type of ML, and do not consider assurance explicitly. Due to space constraints, we discuss these surveys in Appendix A from our online supplementary material.

4 DATA MANAGEMENT

Fundamentally, all ML approaches start with data. These data describe the desired relationship between the ML model inputs and outputs, the latter of which may be implicit for unsupervised approaches. Equivalently, these data encode the requirements we wish to be embodied in our ML model. Consequently, any assurance argument needs to explicitly consider data.

4.1 Inputs and Outputs

The key input artefact to the Data Management stage is the set of requirements that the model is required to satisfy. These may be informed by verification artefacts produced by earlier iterations of the ML lifecycle. The key output artefacts from this stage are datasets: There is a combined dataset that is used by the development team for training and validating the model; there is also a separate verification dataset, which can be used by an independent verification team.

4.2 Activities

4.2.1 Collection. This activity is concerned with collecting data from an originating source. These data may be subsequently enhanced by other activities within the Data Management stage. New data may be collected, or a pre-existing dataset may be re-used (or extended). Data may be obtained from a controlled process, or they may arise from observations of an uncontrolled process: This process may occur in the real world, or it may occur in a synthetic environment.

4.2.2 Preprocessing. For the purposes of this article we assume that preprocessing is a one-to-one mapping: It adjusts each collected (raw) sample in an appropriate manner. It is often concerned with standardising the data in some way, e.g., ensuring all images are of the same size [97]. Manual addition of labels to collected samples is another form of preprocessing.

4.2.3 Augmentation. Augmentation increases the number of samples in a dataset. Typically, new samples are derived from existing samples, so augmentation is, generally, a one-to-many mapping. Augmentation is often used due to the difficulty of collecting observational data (e.g., for reasons of cost or ethics [140]). Augmentation can also be used to help instil certain properties in the trained model, e.g., robustness to adversarial examples [61].

4.2.4 Analysis. Analysis may be required to guide aspects of collection and augmentation (e.g., to ensure there is an appropriate class balance within the dataset). Exploratory analysis is also needed to provide assurance that Data Management artefacts exhibit the desiderata below.

4.3 Desiderata

From an assurance perspective, the datasets produced during the Data Management stage should exhibit the following key properties:

(1) Relevant—This property considers the intersection between the dataset and the desired behaviour in the intended operational domain. For example, a dataset that only included German road signs would not be Relevant for a system intended to operate on UK roads.
(2) Complete—This property considers the way samples are distributed across the input domain and subspaces of it. In particular, it considers whether suitable distributions and combinations of features are present. For example, an image dataset that displayed an inappropriate correlation between image background and type of animal would not be complete [138].
(3) Balanced—This property considers the distribution of features that are included in the dataset. For classification problems, a key consideration is the balance between the number of samples in each class [63]. This property takes an internal perspective; it focuses on the dataset as an abstract entity to which a generic learning algorithm will be applied. In contrast, the Complete property takes an external perspective; it considers the dataset within the intended operational domain.
(4) Accurate—This property considers how measurement (and measurement-like) issues can affect the way that samples reflect the intended operational domain. It covers aspects like sensor accuracy and labelling errors [26]. The correctness of data collection and preprocessing software is also relevant to this property, as is configuration management.

Conceptually, since it relates to real-world behaviour, the Relevant desideratum is concerned with validation. The other three desiderata are concerned with aspects of verification.

4.4 Methods

This section considers each of the four desiderata in turn. Methods that can be applied during each Data Management activity (Table 1), to help achieve the desired key property, are discussed.

4.4.1 Relevant. By definition, a dataset collected during the operational use of the planned system will be relevant. However, this is unlikely to be a practical way of obtaining all required data.

If the approach adopted for data collection involves re-use of a pre-existing dataset, then it needs to be acquired from an appropriate source. Malicious entries in the dataset can introduce a backdoor, which causes the model to behave in an attacker-defined way on specific inputs (or small classes of inputs) [35]. Despite recent research into the detection of backdoors [100, 148, 178], it remains an open challenge (listed as DM01 in Table 2). It follows that pre-existing datasets should be obtained from trustworthy sources via means that provide strong guarantees on integrity during transit.

If the data samples are being collected from controlled trials, then we would require an appropriate experimental plan that justifies the choice of feature values (inputs) included in the trial. If the trial involves real-world observations, then traditional experimental design techniques will be appropriate [87]. Conversely, if the trial is conducted entirely in a simulated environment, then techniques for the design and analysis of computer experiments will be beneficial [146].

If the dataset contains synthetic samples (either from collection or as a result of augmentation), then we would expect evidence that the synthesis process is appropriately representative of the real world. Often, synthesis involves some form of simulation, which ought to be suitably verified and validated [149], as there are examples of ML-based approaches affected by simulation bugs [37]. Demonstrating that synthetic data is appropriate to the real-world intended operational domain, rather than a particular idiosyncrasy of a simulation, is an open challenge (DM02 in Table 2).

A dataset can be made irrelevant by data leakage. This occurs when the training data includes information that will be unavailable to the system within which the ML model will be used [83]. One way of reducing the likelihood of leakage is to only include in the training data features that can "legitimately" be used to infer the required output. For example, patient identifiers are unlikely to be legitimate features for any medical diagnosis system but may have distinctive values for patients already diagnosed with the condition that the ML model is meant to identify [142]. **Exploratory data analysis (EDA)** [171] can help identify potential sources of leakage: a surprising degree of correlation between a feature and an output may be indicative of leakage. That said, detecting and correcting for data leakage is an open challenge (DM03 in Table 2).

Although it appears counterintuitive, augmenting a dataset by including samples that are highly unlikely to be observed during operational use can increase relevance. For classification problems, adversarial inputs [162] are specially crafted inputs that a human would classify correctly but are confidently misclassified by a trained model. Including adversarial inputs *with the correct class* in the training data [125] can help reduce mis-classification and hence increase relevance. Introducing an artificial *unknown*, or "dustbin," class and augmenting the data with suitably placed samples attributed to this class [1] can also help.

Finally, *unwanted bias* (i.e., systematic error ultimately leading to unfair advantage for a privileged class of system users) can significantly impact the relevance of a dataset. This can be addressed using preprocessing techniques that remove the predictability of data features such as ethnicity, age or gender [52] or augmentation techniques that involve data relabelling/reweighing/resampling [80]. It can also be addressed during the Model Learning and Model

Deployment stages. An industry-ready toolkit that implements a range of methods for addressing unwanted bias is available [16].

4.4.2 Complete. Recall that this property is about how the dataset is distributed across the input domain. For the purposes of our discussion, we define four different, but overlapping, spaces related to that domain:

(1) The *input domain* space, $\mathcal{I}$, which is the set of inputs that the model can accept. Equivalently, this set is defined by the input parameters of the software implementation that instantiates the model.
(2) The *operational domain* space, $\mathcal{O} \subset \mathcal{I}$, which is the set of inputs that the model may be expected to receive when used within the intended operational domain.
(3) The *failure domain* space, $\mathcal{F} \subset \mathcal{I}$, which is the set of inputs the model may receive if there are failures elsewhere in the system. The distinction between $\mathcal{F}$ and $\mathcal{O}$ is best conveyed by noting that $\mathcal{F}$ covers system states, whilst $\mathcal{O}$ covers environmental effects: A cracked camera lens should be covered in $\mathcal{F}$; a fly on the lens should be covered in $\mathcal{O}$.
(4) The *adversarial domain* space, $\mathcal{A} \subset \mathcal{I}$, which is the set of inputs the model may receive if it is being attacked by an adversary. This includes adversarial examples, where small changes to an input cause mis-classification [162], as well as more general cyber-related attacks.

The consideration of whether a dataset is complete with regards to the input domain can be informed by simple statistical analysis and EDA [171], supported by discussions with experts from the intended operational domain. Simple plots showing the marginal distribution of each feature can be surprisingly informative. Similarly, the ratio of sampling density between densely sampled and sparsely sampled regions is informative [19] as is, for classification problems, identifying regions that only contain a single class [12]. Identifying any large **empty hyper-rectangles (EHRs)** [98], which are large regions without any samples, is also important. If an operational input comes from the central portion of a large EHR, then, generally speaking, it is appropriate for the system to know the model is working from an area for which no training data were provided.

Shortfalls in completeness across the input domain can be addressed via collection or augmentation. Since a shortfall will relate to a lack of samples in a specific part of the input domain, further collection is most appropriate in the case of controlled trials. The risk of too much data yielding spurious correlations should also be considered [30].

Understanding completeness from the perspective of the *operational domain* space is challenging. Typically, we would expect the input space $\mathcal{I}$ to be high-dimensional, with $\mathcal{O}$ being a much lower-dimensional manifold within that space [150]. Insights into the scope of $\mathcal{O}$ can be obtained by requirements decomposition. The notion of situation coverage [5] generalises these considerations. Lists of factors, based on the operational design domain, as well as object and event detection and response can also be useful [91].

If an increased coverage of $\mathcal{O}$ is needed, then this could be achieved via the use of a **generative adversarial network (GAN)**[3] [10] that has been trained to model the distributions of each class.

Although the preceding paragraphs have surveyed multiple methods, understanding completeness across the operational domain remains an open challenge (DM04 in Table 2).

Completeness across the $\mathcal{F}$ space can be understood by systematically examining the system architecture to identify failures that could affect the model's input. Note that the system

[3] A GAN is a network specifically designed to provide inputs for another network. A classification network tries to learn the boundary between classes, whereas a GAN tries to learn the distribution of individual classes.

architecture may protect the model from the effects of some failures: For example, the system may not present the model with images from a camera that has failed its built-in test. In some cases it may be possible to collect samples that relate to particular failures. However, for reasons of cost, practicality and safety, augmentation is likely to be needed to achieve suitable completeness of this space [6]. Finding verifiable ways of achieving this augmentation is an open challenge (DM05 in Table 2).

Understanding completeness across the adversarial domain $\mathcal{A}$ involves checking: the model's susceptibility to known ways of generating adversarial examples [61, 117, 162, 188] and its behaviour when presented with inputs crafted to achieve some other form of behaviour, for example, a not-a-number error. Whilst they are useful, both of these methods are subject to the "unknown unknowns" problem. More generally, demonstrating completeness across the adversarial domain is an open challenge (DM06 in Table 2).

4.4.3 Balanced. This property is concerned with the distribution of the dataset, viewed from an internal perspective. Initially, it is easiest to think about balance solely from the perspective of supervised classification, where a key consideration is the number of samples in each class. If this is unbalanced, then simple measures of performance (e.g., classifier accuracy) may be insufficient [101].

As it only involves counting the number of samples in each class, detecting class imbalance is straightforward. Its effects can be countered in several ways; for example, in the Model Learning and Model Verification stages, performance measures can be class-specific, or weighted to account for class imbalance [63]. Importance weighting can, however, be ineffective for deep networks trained for many epochs [28]. Alternatively, or additionally, in the Data Management stage augmentation can be used to correct (or reduce) the class imbalance, either by oversampling the minority class, or by undersampling the majority class, or using a combination of these approaches[4] [101]. If data are being collected from a controlled trial, then another approach to addressing class imbalance is to perform additional collection, targeted toward the minority class.

Class imbalance can be viewed as being a special case of rarity [182]. Another way rarity can manifest is through small disjuncts, which are small, isolated regions of the input domain that contain a single class. Analysis of single-class regions [12] can inform the search for small disjuncts, as can EDA and expertise in the intended operational domain. Nevertheless, finding small disjuncts remains an open challenge (DM07 in Table 2).

Class imbalance can also be viewed as a special case of a phenomenon that applies to all ML approaches: feature imbalance. Suppose we wish to create a model that applies to people of all ages. If almost all of our data relate to people between the ages of 20 and 40, then we have an imbalance in this feature. Situations like this are not atypical when data are collected from volunteers. Detecting such imbalances is straightforward; understanding their influence on model behaviour and correcting for them are both open challenges (DM08 and DM09 in Table 2).

4.4.4 Accurate. Recall that this property is concerned with measurement (and measurement-like) issues. If sensors are used to record information as part of data collection, then both sensor precision and accuracy need to be considered. If either of these is high, then there may be benefit in augmenting the collected data with samples drawn from a distribution that reflects precision or accuracy errors.

For supervised learning, accuracy of labels is an important issue. When labelling is being conducted by a large group of people, using a form of crowdsourcing, there are two key issues: quality

[4] Note that the last two approaches involve removing samples (corresponding to the majority class) from the dataset; this differs from the normal view whereby augmentation increases the number of samples in the dataset.

control in task processing and postprocessing to improve data quality [155]. For some especially sensitive applications, performing all labelling using a comparatively small number of in-house staff may be preferable [164]. Regardless of who performs the labelling, the actual value of a feature is often unambiguously defined [157]. However, in some cases this may not be possible: for example, is a person walking astride a bicycle a pedestrian or a cyclist? In some cases, ambiguity may be unimportant; in others it might be critical. Consequently, labelling discrepancies are likely, especially when labels are generated by humans. Preventing, detecting and resolving these discrepancies is an open challenge (DM10 in Table 2).

The data collection process should generally be documented in a way that accounts for potential weaknesses in the approach. If the process uses manual recording of information, then we would expect steps to be taken to ensure attention does not waver and records are accurate. Conversely, if the data collection process uses logging software, then confidence that this software is behaving as expected should be obtained, e.g., using traditional approaches for software quality [75, 143].

This notion of correct behaviour applies across all software used in the Data Management stage (and to all software used in the ML lifecycle). Data collection software may be relatively simple, merely consisting of an automatic recording of sensed values. Alternatively, it may be very complex, involving a highly-realistic simulation of the intended operational domain. The amount of evidence needed to demonstrate correct behaviour is related to the complexity of the software. Providing sufficient evidence for a complex simulation is an open challenge (DM11 in Table 2).

Given their importance, datasets should be protected against unintentional and unauthorised changes. Methods used in traditional software development (e.g., References [75, 143]) may be appropriate for this task, but they may be challenged by the large volume and by the non-textual nature of many of the datasets used in ML.

4.5 Summary and Open Challenges

Table 1 summarises the assurance methods that can be applied during the Data Management stage. For ease of reference, the methods are presented in the order they were introduced in the preceding discussion. Methods are also matched to activities and desiderata.

Table 1 shows that there are relatively few methods associated with the preprocessing activity. This may be because preprocessing is, inevitably, problem-specific. Likewise, there are few methods associated with the Accurate desideratum. This may reflect the widespread use of commonly available datasets (e.g., ImageNet [40] and MNIST) within the research literature, which de-emphasises issues associated with data collection and curation, like Accuracy.

Open challenges associated with the Data Management stage are shown in Table 2. The relevance and nature of these challenges have been established earlier in this section. For ease of reference, each challenge is matched to the artefact desideratum that it is most closely related to. It is apparent that, with the exception of understanding the effect of feature imbalance on model performance, these open challenges do *not* relate to the core process of learning a model. As such, they emphasise important areas that are insufficiently covered in the research literature. Examples include being able to demonstrate: that the model is sufficiently secure—from a cyber perspective (open challenge DM01); that the data are fit-for-purpose (DM02, DM03); that the data cover operational, failure and adversarial domains (DM04, DM05, DM06); that the data are balanced, across and within classes (DM07, DM08, DM09); that manual data collection has not been compromised (DM10); and that simulations are suitably detailed and representative of the real world (DM11).

5 MODEL LEARNING

The Model Learning stage of the ML lifecycle is concerned with creating a model, or algorithm, from the data presented to it. A good model will replicate the desired relationship between inputs

Table 1. Assurance Methods for the Data Management Stage

	Associated activities[†]				Supported desiderata[‡]			
Method	Collection	Preprocess.	Augment.	Analysis	Relevant	Complete	Balanced	Accurate
Use trusted data sources, with data-transit integrity guarantees	✔				★			
Experimental design [87, 146]	✔		✔		★	★	☆	
Simulation verification and validation [149]			✔		★	☆	☆	
Exploratory data analysis [171]				✔		★	★	
Use adversarial examples [125]			✔		☆	★		
Include a “dustbin” class [1]			✔		☆	★		
Remove unwanted bias [16]		✔	✔		★		☆	
Compare sampling density [19]			✓	✔		★	☆	
Identify empty and single-class regions [12, 98]			✓	✔		★	☆	
Use situation coverage [5]				✔		★		
Examine system failure cases				✔		★		
Oversampling & undersampling [101]				✔		★	★	
Check for within-class [76] and feature imbalance				✔		★		
Use a GAN [10]			✔			★	☆	
Augment data to account for sensor errors	✓		✔		☆			★
Confirm correct software behaviour [75, 143]	✓	✔	✔	✓	☆	★	☆	☆
Use documented processes	✔	✔	✔	✔	☆			★
Apply configuration management [75, 143]	✔	✔	✔	✔	☆			★

[†] ✔ = activity that the method is typically used in; ✓ = activity that may use the method.
[‡] ★ = desideratum supported by the method; ☆ = desideratum partly supported by the method.

Table 2. Open Challenges for the Assurance Concerns Associated with the Data Management (DM) Stage

ID	**Open Challenge**	**Desideratum (Section)**
DM01	Detecting backdoors in data	Relevant (Section 4.4.1)
DM02	Demonstrating synthetic data appropriateness to the operational domain	
DM03	Detecting and correcting for data leakage	
DM04	Measuring completeness with respect to the operational domain	Complete (Section 4.4.2)
DM05	Deriving ways of drawing samples from the failure domain	
DM06	Measuring completeness with respect to the adversarial domain	
DM07	Finding small disjuncts, especially for within-class imbalances	Balanced (Section 4.4.3)
DM08	Understanding the effect of feature imbalance on model performance	
DM09	Correcting for feature imbalance	
DM10	Maintaining consistency across multiple human collectors/preprocessors	Accurate (Section 4.4.4)
DM11	Verifying the accuracy of a complex simulation	

and outputs present in the training set, and will satisfy non-functional requirements such as providing an output within a given time and using an acceptable amount of computational resources.

5.1 Inputs and Outputs

The key input artefact to this stage is the training dataset produced by the Data Management stage. The key output artefacts are a machine learned model for verification in the next stage of the ML lifecycle and a performance deficit report used to inform remedial data management activities.

5.2 Activities

5.2.1 Model Selection. This activity decides the model type, variant and, where applicable, the structure of the model to be produced in the Model Learning stage. Numerous types of ML models are available [113, 152], including multiple types of *classification* models (which identify the category that the input belongs to), *regression* models (which predict a continuous-valued attribute), *clustering* models (which group similar items into sets), and *reinforcement learning* models (which provide an optimal set of actions, i.e., a policy, for solving, for instance, a navigation or planning problem).

5.2.2 Training. This activity optimises the performance of the ML model with respect to an objective function that reflects the requirements for the model. To this end, a subset of the training data is used to find internal model parameters (e.g., the weights of a neural network, or the coefficients of a polynomial) that minimise an error metric for the given dataset. The remaining data (i.e, the validation set) are then used to assess the ability of the model to generalise. These two steps are typically iterated many times, with the training hyperparameters tuned between iterations so as to further improve the performance of the model.

5.2.3 Hyperparameter Selection. This activity is concerned with selecting the parameters associated with the training activity, i.e., the hyperparameters. Hyperparameters control the effectiveness of the training process, and ultimately the performance of the resulting model [130]. They are so critical to the success of the ML model that they are often deemed confidential for models used in proprietary systems [177]. There is no clear consensus on how the hyperparameters should be tuned [102]. Typical options include: initialisation with values offered by ML frameworks, manual configuration based on recommendations from literature or experience, or trial and error [130]. Alternatively, the tuning of the hyperparameters can itself be seen as a machine learning task [71, 187].

5.2.4 Transfer Learning. The training of complex models may require weeks of computation on many GPUs [62]. As such, there are clear benefits in reusing ML models across multiple domains. Even when a model cannot be transferred between domains directly, one model may provide a starting point for training a second model, significantly reducing the training time. The activity concerned with reusing models in this way is termed transfer learning [60].

5.3 Desiderata

From an assurance viewpoint, the models generated by the Model Learning stage should exhibit some or all of the key properties described below:

(1) Performant—This property considers quantitative performance metrics applied to the model when deployed within a system. These metrics include traditional ML metrics such as classification accuracy, the **receiver operator characteristic (ROC)** and mean squared error, as well as metrics that consider the system and environment into which the models are deployed.

(2) Robust—This property considers the model's ability to perform well in circumstances where the inputs encountered at runtime are different to those present in the training data. Robustness may be considered with respect to environmental uncertainty, e.g., flooded roads, and system-level variability, e.g., sensor failure, i.e., from the general perspective used in formal verification rather than its ML interpretation as the ability of a model to generalise to data not encountered in training [20, 192].

(3) Reusable—This property considers the ability of a model, or of components of a model, to be reused in systems for which they were not originally intended. For example, a neural network trained for facial recognition in an authentication system may have features that can be reused to identify operator fatigue. More generally, the reuse of pre-trained or commodity off-the-shelf ML models in transfer learning can significantly speed up model learning [108].

(4) Interpretable—This property considers the extent to which the model can produce artefacts that support the analysis of its output, and thus of any decisions based on it. For example, a decision tree may support the production of a narrative explaining the decision to hand over control to a human operator.

5.4 Methods

This section considers each of the four desiderata in turn. Methods applicable during each of the Model Learning activities (Table 3), to help achieve each of the desired properties, are discussed.

5.4.1 Performant. An ML model is performant if it operates as expected according to a measure (or set of measures) that captures relevant characteristics of the model output. Many machine learning problems are phrased in terms of objective functions to be optimized [174], and measures constructed with respect to these objective functions allow models to be compared. Such measures have underpinning assumptions and limitations that should be fully understood before they are used to select a model for deployment in a safety-critical system.

The prediction error of a model has three components: *irreducible error*, which cannot be eliminated regardless of the algorithm or training methods employed; *bias error*, due to simplifying assumptions intended to make learning the model easier; and *variance error*, an estimate of how much the model output would vary if different data were used in the training process. The aim of training is to minimise the bias and variance errors, and therefore the objective functions reflect these errors. The objective functions may also contain simplifying assumptions to aid optimization, and these assumptions must not be present when assessing model performance [59].

Performance measures for classifiers, including accuracy, precision, recall (sensitivity) and specificity, are often derived from their confusion matrix [59, 120, 158]. Comparing models is not always straightforward, with different models showing superior performance against different measures. Composite metrics [59, 158] allow for a tradeoff between measures during the training process. The understanding of evaluation measures has improved over the past two decades but areas where understanding is lacking still exist [54]. While using a single, scalar measure simplifies the selection of a "best" model and is a common practice [46], the ease with which such performance measures can be produced has led to over-reporting of simple metrics without an explicit statement of their relevance to the operating domain. Ensuring that reported measures convey sufficient contextually relevant information remains an open challenge (challenge ML01 from Table 4).

Aggregated measures cannot evaluate models effectively except in the simplest scenarios, and the operating environment influences the required tradeoff between performance metrics. The ROC curve [132] allows for Pareto-optimal model selection using a tradeoff between the true and false positive rates, while the area under the ROC curve [24] assesses the sensitivity of models to changes in operating conditions. Cost curves [46] allow weights to be associated with true and false positives to reflect their importance in the operating domain. Where a single classifier cannot provide an acceptable tradeoff, models identified using the ROC curve may be combined to produce a classifier with better performance than any single model, under real-world operating conditions [131]. This requires tradeoffs to be decided at training time, which is unfeasible for dynamic environments and multi-objective optimisation problems. Developing methods to defer this decision until runtime is an open challenge (ML02 in Table 4).

Whilst the measures presented thus far give an indication of the performance of the model against datasets, they do not encapsulate the users' trust in a model for a specific, possibly rare, operating point. The **intuitive certainty measure (ICM)** [172] is a mechanism to produce an estimate of how certain an ML model is for a specific output based on errors made in the past. ICM compares current and previous sensor data to assess similarity, using previous outcomes for similar environmental conditions to inform trust measures. Due to the probabilistic nature of machine learning [120], models may also be evaluated using classical statistical methods. These methods can answer several key questions [114]: (i) Given the observed accuracy of a model, how well is it likely to estimate unseen samples?, (ii) If a model outperforms another for a specific dataset, how likely is it to be more accurate in general?, and (iii) What is the best way to learn a hypothesis from limited data?

Methods are also available to improve the performance of the ML models. Ensemble learning [147] combines multiple models to produce a model whose performance is superior to that of any of its constituent models. The aggregation of models leads to lower overall bias and to a reduction in variance errors [59]. Bagging and boosting [145] can improve the performance of ensemble models further. Bagging increases model diversity by selecting data subsets for the training of each model in the ensemble. After an individual model is created, boosting identifies the samples for which the model performance is deficient, and increases the likelihood of these samples being selected for subsequent model training. AdaBoost [57], short for Adaptive Boosting, is a widely used boosting algorithm reported to have solved many problems of earlier boosting algorithms [56]. Where the training data are imbalanced, the SMOTE boosting algorithm [33] may be employed.

The selection and optimization of hyperparameters [59] has a significant impact on the performance of models [177]. Given the large number of hyperparameters, tuning them manually is typically unfeasible. Automated optimization strategies are employed instead [71], using methods that include grid search, random search and latin hypercube sampling [89]. Evolutionary algorithms may also be employed for high-dimensional hyperparameter spaces [187]. Selecting the most appropriate method for hyperparameter tuning in a given context and understanding the interaction between hyperparameters and model performance [102] represent open challenges (ML03 and ML04 in Table 4, respectively). Furthermore, there are no guarantees that a tuning strategy will continue to be optimal as the model and data on which it is trained evolve.

We have highlighted multiple metrics for the assessment of model performance, as well as methods that promise improved model performance with respect to these metrics. The selection of appropriate metrics and methods remains a complex task, however, and will depend greatly on the context into which the ML component is to be deployed. Given this complexity and the importance of model performance, the rationale that underpins this selection process should be clearly justified in the resulting assurance case.

5.4.2 *Robust.*

Training optimizes models with respect to an objective function using the data in the training set. The aim of the model learning process, however, is to produce a model that generalises to data not present in the training set but that may be encountered in operation.

Increasing model complexity generally reduces training errors, but noise in the training data may result in overfitting and in a failure of the model to generalise to real-world data. When choosing between competing models one method is then to prefer simple models (Ockham's razor) [145]. An estimate of the model's ability to generalise may be obtained by using k-fold cross-validation [60]. This method partitions the training data into k non-overlapping subsets, with $k-1$ subsets used for training and the remaining subset used for validation. The process is repeated k times, with a different validation subset used each time, and an overall error is calculated as the mean error over the k trials. Methods to avoid overfitting include gathering more training data, reducing the noise present in the training set, and simplifying the model [59].

Data augmentation (Section 4.2.3) can improve the quality of training data and improve robustness of models [88]. Applying transformations to images in the input space may produce models that are robust to changes in the position and orientation of objects in the input space [59] whilst photometric augmentation may increase robustness with respect to lighting and colour [165]. For models of speech, altering the speed of playback for the training set can increase model robustness [88]. Identifying the best augmentation methods for a given context can be difficult, and Antoniou et al. [10] propose an automated augmentation method that uses generative adversarial networks to augment datasets without reference to a contextual setting. These methods require the identification of all possible deviations from the training dataset, so that deficiencies can be compensated through augmentation. Assuring the completeness of (augmented) datasets has already been identified as an open challenge (DM02 in Table 2). Even when a dataset is complete, the practice of reporting aggregated generalisation errors means that assessing the impact of each type of deviation on model performance is challenging. Indeed, the complexity of the open environments in which many critical systems operate, and for which assurance cases are required, means that decoupling the effects of different perturbations on model performance remains an open challenge (ML05 in Table 4).

Regularization methods are intended to reduce a model's generalization error but not its training error [60, 145], e.g., by augmenting the objective function with a term that penalises model complexity. The ℓ_0, ℓ_1, or ℓ_2 norm are commonly used [120], with the term chosen based on the learning context and model type. The ridge regression [59] method may be used for models with low bias and high variance. This method adds a weighted term to the objective function that aims to keep the weights internal to the model as low as possible. Early stopping [129] is a simple method that avoids overfitting by stopping the training if the validation error begins to rise. For deep neural networks, dropout [67, 159] is the most popular regularization method. Dropout selects different subsets of neurons to be ignored at each training step. This makes the model less reliant on any one neuron, and hence increases its robustness. Dropconnect [176] employs a similar technique to improve the robustness of large networks by setting subsets of weights in fully connected layers to zero. For image classification tasks, randomly erasing portions of input images can increase the robustness of the generated models [193] by ensuring that the model is not overly reliant on any particular subset of the training data.

Robustness with respect to adversarial perturbations for image classification is problematic for deep neural networks, even when the perturbations are imperceptible to humans [162] or the model is robust to random noise [51]. Whilst initially deemed a consequence of the high non-linearity of neural networks, recent results suggest that the "success" of adversarial examples is due to the low flexibility of classifies, and affects classification models more widely [50]. Adversarial robustness may therefore be considered as a measure of the *distinguishability* of a classifier.

Ross and Doshi-Velez [141] introduced a batch normalization method that penalises parameter sensitivity to increase robustness to adversarial examples. This method adds noise to the hidden units of a deep neural network at training time, can have a regularization effect, and sometimes makes dropout unnecessary [60]. Although regularization can improve model robustness without knowledge of the possible deviations from the training dataset, understanding the nature of robustness in a contextually meaningful manner remains an open challenge (ML06 in Table 4).

5.4.3 Reusable. Machine learning is typically computationally expensive, and repurposing models from related domains can reduce the cost of training new models. Transfer learning [183] allows for a model learned in one domain to be exploited in a second domain, as long as the domains are similar enough so that features learned in the source domain are applicable to the target domain. Where this is the case, all or part of a model may be transferred to reduce the training cost.

Convolutional neural networks (CNN) are particularly suited for partial model transfer [59] since the convolutional layers encode features in the input space, whilst the fully connected layers encode reasoning based on those features. Thus, a CNN trained on human faces is likely to have feature extraction capabilities to recognise eyes, noses, and so on. To train a CNN from scratch for a classifier that considers human faces is wasteful if a CNN for similar tasks already exists. By taking the convolutional layers from a source model and learning a new set of weights for the fully connected set of layers, training times may be significantly reduced [69, 124]. Similarly, transfer learning has been shown to be effective for random forests [153, 160], where subsets of trees can be reused. More generally, the identification of "similar" operational contexts is difficult, and defining a meaningful similarity measure in high-dimensional spaces is an open challenge (ML07 in Table 4).

Even with significant differences between the source and target domains, an existing model may be valuable. Initialising the parameters of a model to be learned using values obtained in a similar domain may greatly reduce training times, as shown by the successful use of transfer learning in the classification of sentiments, human activities, software defects, and multi-language texts [183].

Using pre-existing models as the starting point for a new problem can be effective. As such, it may be desirable to make use of models previously used to tackle problems in related domains. A growing number of *model zoos* [59] containing such models are being set up by many core learning technology platforms [116], as well as by researchers and engineers [115].

Transfer learning resembles software component reuse, and may allow the reuse of assurance evidence about ML models across domains, as long as the assumptions surrounding the assurance are also transferable between the source and target domains. However, a key aspect in the assurance of components is that they should be reproducible and, at least for complex (deep) ML models, reproducing the learning process is rarely straightforward [174]. Indeed, reproducing ML results requires significant configuration management, which is often overlooked by ML teams [189].

Another reason for caution when adopting third-party model structures, weights and processes is that transfer learning can also transfer failures and faults from the source to the target domain [62]. Indeed, ensuring that existing models are free from faults is an open challenge (ML08 in Table 4).

5.4.4 Interpretable. For many critical domains where assurance is required, it is essential that ML models are interpretable. "Interpretable" and "explainable" are closely related concepts, with "interpretable" used in the ML community and "explainable" preferred in the AI community [2]. We use the term "interpretable" when referring to properties of machine learned models, and "explainable" when systems features and contexts of use are considered.

Interpretable models aid assurance by providing evidence that allows for [2, 96, 99]: justifying the results provided by a model, supporting the identification and correction of errors, aiding model improvement, and providing insight with respect to the operational domain.

The difficulty of providing interpretable models stems from the frequent use of complex ML models whose structure and size makes it impossible for a human to construct a mental model that can explain the features and parameters of the model in a contextually meaningful manner. As such, one approach to producing interpretable models is to reduce model complexity to a level where the number of features and parameters are no longer a barrier to understanding. However, this is likely to also reduce model accuracy, and therefore a better approach is to use *inherently interpretable models*, i.e., models that satisfy domain-specific constraints that may include monotonicity, additivity or causality [144].

Methods that aid in the production of interpretable models can be classified by the scope of the explanations they generate. Global methods generate evidence that apply to a whole model, and support design and assurance activities by allowing reasoning about all possible future outcomes for the model. Local methods generate explanations for an individual decision, and may be used to analyse why a particular problem occurred, and to improve the model so future events of this type are avoided. Methods can also be classified as model-agnostic and model-specific [2]. Model-agnostic methods are mostly applicable post-hoc (after training), and include providing natural language explanations [95], using model visualisations to support understanding [106], and explaining by example [3]. Much less common, model-specific methods [2] typically provide more detailed explanations, but restrict the users' choice of model, and therefore are only suited if the limitations of the model(s) they can work with are acceptable.

Despite significant research into interpretable models, there are no global methods providing contextually relevant insights to aid human understanding for complex ML models (ML09 in Table 4). In addition, although several post-hoc local methods exist, there is no systematic approach to infer global properties of the model from local cases. Without such methods, interpretable models cannot aid structural model improvements and error correction at a global level (ML10 in Table 4).

5.5 Summary and Open Challenges

Table 3 summaries the assurance methods applicable during the Model Learning stage. The methods are presented in the order that they are introduced in the preceding discussion, and are matched to the activities with which they are associated and to the desiderata that they support. The majority of these methods focus on the performance and robustness of ML models. Model reusability and interpretability are only supported by a few methods that typically restrict the types of model that can be used. This imbalance reflects the different maturity of the research on the four desiderata, with the need for reuse and interpretability arising more prominently after the recent advances in deep learning and increases in the complexity of ML models.

Open challenges for the assurance of the Model Learning stage are presented in Table 4, organised into categories based on the most relevant desideratum for each challenge. The importance and nature of these challenges have been established earlier in this section. A common theme across many of these challenges is the need for integrating concerns associated with the operating context into the Machine Learning stage (open challenges ML01, ML03, ML05, ML06, ML07). Open challenges also exist in the evaluation of performance of models with respect to multi-objective evaluation criteria (ML02); understanding the link between model performance and hyperparameter selection (ML04) and ensuring that where transfer learning is adopted that existing models are free from errors (ML08). While there has been a great deal of research focused on interpretable models, methods that apply globally to complex models (ML09) are still lacking. Where local explanations are provided, methods are needed to extract global model properties from them (ML10).

Table 3. Assurance Methods for the Model Learning Stage

	Associated activities[†]				Supported desiderata[‡]			
	Model	Training	Hyperparam.	Transfer				
Method	Selection		Selection	Learning	Performant	Robust	Reusable	Interpretable
Use appropriate performance measures [54, 174]	✓	✔			★	★		
Statistical tests [114, 120]	✓	✔			★			
Ensemble Learning [147]	✔	✔		✔	★	★		
Optimise hyperparameters [71, 187]		✔	✔		★	★		
Batch Normalization [73]		✔	✔		★	★		
Prefer simpler models [3, 145]	✔	✓			☆	★		☆
Augment training data		✔			★	★		
Regularization methods [59]		✔	✔			★		
Use early stopping		✔	✔			★		
Use models that intrinsically support reuse [2]	✔			✔			★	☆
Transfer Learning [183]	✔	✓		✔			★	☆
Use model zoos [59]	✔	✓		✔			★	
Post-hoc interpretability methods [3, 95, 106]		✔						★

[†] ✔ = activity that the method is typically used in; ✓ = activity that may use the method.
[‡] ★ = desideratum supported by the method; ☆ = desideratum partly supported by the method.

Table 4. Open Challenges for the Assurance Concerns Associated with the Model Learning (ML) Stage

ID	Open Challenge	Desideratum (Section)
ML01	Selecting measures that represent operational context	Performant (Section 5.4.1)
ML02	Multi-objective performance evaluation at runtime	
ML03	Using operational context to inform hyperparameter-tuning strategies	
ML04	Understanding the impact of hyperparameters on model performance	
ML05	Decoupling the effects of perturbations in the input space	Robust (Section 5.4.2)
ML06	Inferring contextual robustness from evaluation metrics	
ML07	Identifying similarity in operational contexts	Reusable (Section 5.4.3)
ML08	Ensuring existing models are free from faults	
ML09	Global methods for interpretability in complex models	Interpretable (Section 5.4.4)
ML10	Inferring global model properties from local cases	

6 MODEL VERIFICATION

The Model Verification stage of the ML lifecycle is concerned with the provision of auditable evidence that a model will continue to satisfy its requirements when exposed to inputs that are not present in the training data.

6.1 Inputs and Outputs

The key input artefact to this stage is the trained model produced by the Model Learning stage. The key output artefacts are a verified model, and a verification result that provides sufficient information to allow potential users to determine if the model is suitable for its intended application(s).

6.2 Activities

6.2.1 Requirement Encoding. This activity involves transforming requirements into both tests and mathematical properties, where the latter can be verified using formal techniques. Requirements encoding requires a knowledge of the application domain, such that the intent that is implicit in the requirements may be encoded as explicit tests and properties. A knowledge of the technology that underpins the model is also required, such that technology-specific issues may be assessed through the creation of appropriate tests and properties.

6.2.2 Test-based Verification. This activity involves providing test cases (i.e., specially formed inputs or sequences of inputs) to the trained model and checking the outputs against predefined expected results. A large part of this activity involves an independent examination of properties considered during the Model Learning stage (cf. Section 5), especially those related to the Performant and Robust desiderata. In addition, this activity also considers test completeness, i.e., whether the set of tests exercised the model and covered its input domain sufficiently. The latter objective is directly related to the Complete desideratum from the Data Management stage (cf. Section 4). Finally, tests should be repeatable, allowing for the systematic elimination of errors and the identification of improvement between development cycles.

6.2.3 Formal Verification. This activity involves the use of mathematical techniques to provide irrefutable evidence that the model satisfies formally specified properties derived from its requirements. Counterexamples are typically provided for properties that are violated, and can be used to inform further iterations of activities from the Data Management and Model Learning stages.

6.3 Desiderata

To be compelling, the verification results (i.e., the evidence) generated by the Model Verification stage should exhibit the following key properties:

(1) Comprehensive—This property is concerned with the ability of Model Verification to cover: (i) all the requirements and operating conditions associated with the intended use of the model; and (ii) all the desiderata from the previous stages of the ML lifecycle (e.g., the completeness of the training data, and the robustness of the model).
(2) Contextually Relevant—This desideratum considers the extent to which test cases and formally verified properties can be mapped to contextually meaningful aspects of the system that will use the model. For example, for a model used in an autonomous car, robustness with respect to image contrast is less meaningful than robustness to variation in weather conditions.
(3) Comprehensible—This property considers the extent to which verification results can be understood by those using them in activities ranging from data preparation and model development to system development and regulatory approval. A clear link should exist between the aim of the Model Verification and the guarantees it provides. Limitations and assumptions should be clearly identified, and results that show requirement violations should convey sufficient information to allow the underlying cause(s) for the violations to be fixed.

6.4 Methods

6.4.1 Comprehensive. Compared to traditional software the dimension and testing space of an ML model is potentially much larger [25]. Ensuring that model verification is comprehensive requires a systematic approach to identify faults due to conceptual misunderstandings and faults introduced during the Data Management and Model Learning activities (Table 5).

Conceptual misunderstandings may occur during the construction of requirements. They impact both Data Management and Model Learning, and may lead to errors that include: data that are not representative of the intended operational environment; loss functions that do not capture the original intent; and design tradeoffs that detrimentally affect performance and robustness when deployed in real-world contexts. Independent consideration of requirements is important in traditional software but, it could be argued, it is even more important for ML because the associated workflow includes no formal, traceable hierarchical requirements decomposition [13].

Traditional approaches to safety-critical software development distinguish between normal testing and robustness testing [143]. The former is concerned with typical behaviour, whilst the latter tries to induce undesirable behaviour on the part of the software. Stress testing, especially that specifically designed for autonomy software, is a valuable approach [70]. In terms of the spaces discussed in Section 4, normal testing tends to focus on the operational domain, $\mathcal{O}$; it can also include aspects of the failure domain, $\mathcal{F}$, and the adversarial domain, $\mathcal{A}$. Conversely, robustness testing utilises the entire input domain, $\mathcal{I}$, including, but not limited to, elements of $\mathcal{F}$ and $\mathcal{A}$. Robustness testing for traditional software is informed by decades of accumulated knowledge on typical errors (e.g., numeric overflow and buffer overruns). Whilst a few typical errors have also been identified for ML (e.g., overfitting and backdoors [35]), the knowledge about such errors is limited and rarely accompanied by an understanding of how these errors may be detected and corrected. Developing this knowledge is an open challenge (challenge MV01 in Table 6).

Coverage is an important measure for assessing the comprehensiveness of software testing. For traditional software, coverage focuses on the structure of the software. For example, statement coverage or branch coverage can be used as a surrogate for measuring how much of the software's behaviour has been tested. However, measuring ML model coverage in the same way is not informative: achieving high branch coverage for the code that implements a neuron activation function tells little, if anything, about the behaviour of the trained network. For ML, test coverage needs to be considered from the perspectives of both data and model structure. The methods associated with the Complete desiderata from the Data Management stage can inform data coverage. In addition, model-related coverage methods have been proposed in recent years [104, 126, 161], although achieving high coverage is generally unfeasible for large models due to the high dimensionality of their input and feature spaces. Traditional software testing employs combinatorial testing to mitigate this problem, and DeepCT [103] provides combinatorial testing for deep-learning models.

However, whilst these methods provide a means of *measuring* coverage, the benefits of achieving a particular level of coverage are not clear. Put another way, we understand the theoretical value of increased data coverage, but its empirical utility has not been demonstrated. As such, it is impossible to define coverage thresholds that should be achieved. Indeed, it is unclear whether a generic threshold is appropriate, or whether coverage thresholds are inevitably application specific. Consequently, deriving a set of coverage measures that address both data and model structure, and demonstrating their practical utility remains an open challenge (MV02 in Table 6).

The susceptibility of neural networks to adversarial examples is well known, and can be mitigated using formal verification methods. These methods ensure local adversarial robustness by providing mathematical guarantees that, for a suitably-sized region around an input-space point, the same decision will always be returned [68]. This is an active research area in which tools have been developed using satisfiability and **satisfiability modulo theory (SMT)** techniques, mixed-integer linear programming and geometric reachability [47, 48, 58, 81]. While scalability was initially an issue, more recent work has provided tool sets and frameworks that can handle larger networks, as well as networks that make use of more diverse structures, such as max pooling [43, 82, 169, 170]. Formal verification methods are all reliant on the assumptions of proximity and smoothness. Proximity concerns the notion that two similar inputs will have similar outputs, while smoothness assumes that the model smoothly transitions between values [173]. However, without an understanding of the model's context, it is difficult to ascertain whether two inputs are similar (e.g., based on a meaningful distance metric), or to challenge smoothness assumptions when discontinuities are present in the modelled domain.

While many ML formal verification methods focus on neural networks, several techniques have been proposed for different ML paradigms, including random forests [7, 168, 181], nearest neighbours [180], Bayesian networks classifiers [156], and support vector machines [136]. Test-based verification may include use of a simulation to generate test cases. In this case, appropriate confidence needs to be placed in the simulation. For normal testing, the simulation-related concepts discussed in Section 4 (e.g., verification and validation) are relevant. If the term "simulation" is interpreted widely, then robustness testing could include simulations that produce pseudo-random inputs (i.e., fuzzing), or simulations that try to invoke certain paths within the model (i.e., guided fuzzing [123]). In these cases, we need confidence that the "simulation" is working as intended (verification), but we do not need it to be representative of the real world (validation).

Last, the model verification may also need to be applied to any ML libraries or platforms used in the Model Learning stage for a number of safety-critical systems. Errors in this software are difficult to identify, as the iterative nature of model training and parameter tuning can mask software implementation errors. Proving the correctness of ML libraries and platforms requires program verification techniques to be applied [154].

6.4.2 Contextually Relevant. Requirement encoding should consider how the tests and formal properties constructed for verification map to context. This is particularly difficult for high-dimensional problems such as those tackled using deep neural networks. Verification methods that assess model performance with respect to proximity and smoothness are mathematically provable, but defining regions around a point in space does little to indicate the types of real-world perturbation that can, or cannot, be tolerated by the system (and those that are likely, or unlikely, to occur in reality). As such, mapping requirements to model features is an open challenge (MV04 in Table 6).

Depending on the intended application, Model Verification may need to explicitly consider unwanted bias. In particular, if the context of model use includes a legally protected characteristic (e.g., age, race or gender) then considering bias is a necessity. As discussed in Section 4.4.1, there are several ways this can be achieved, and an industry-ready toolkit is available [16].

Contextually relevant verification methods such as DeepTest [167] and DeepRoad [190] have been developed for autonomous driving. DeepTest employs neuron coverage to guide the generation of tests cases for ML models used in this application domain. Test cases are constructed as contextually relevant transformations of the dataset, e.g., by adding synthetic but realistic fog and camera lens distortion to images. DeepTest leverages principles of metamorphic testing, so that even when the valid output for a set of inputs is unknown it can be inferred from similar cases (e.g., an image with and without camera lens distortion should return the same result). DeepRoad works in a similar way, but generates its contextually relevant images using a generative adversarial network. These methods work for neural networks used in autonomous driving, but developing a general framework for synthesizing test data for other contexts is an open challenge (MV05 in Table 6).

Although adversarial examples are widely used to verify the robustness of neural networks, they typically disregard the semantics and context of the system into which the ML model will be deployed. Semantic adversarial deep learning [45] is a method that avoids this limitation through considering the model context explicitly, first by using input modifications informed by contextual semantics (much like DeepTest and DeepRoad), and second by using system specifications to assess the system-level impact of invalid model outputs. By identifying model errors that lead to system failures, the latter technique aids the model repair and re-design.

The verification of reinforcement learning [173] requires a number of different features to be considered. When **Markov decision process (MDP)** models of the environment are devised by domain experts, the MDP states are nominally associated with contextually-relevant operating states. As such, systems requirements can be encoded as properties in temporal logics and probabilistic model checkers may be used to provide probabilistic performance guarantees [109]. When these models are learned from data, it is difficult to map model states to real-world contexts, and constructing meaningful properties is an open challenge (MV06 in Table 6).

6.4.3 Comprehensible. The utility of Model Verification is enhanced if its results provide information that aids the fixing of any errors identified by the test-based and formal verification of ML models. One method that supports the generation of comprehensible verification results is to use contextually relevant testing criteria, as previously discussed. Another method is to use counterexample-guided data augmentation [44]. For traditional software, the counterexamples provided by formal verification guide the eradication of errors by pointing to a particular block of code or execution sequence. For ML models, counterexamples are more difficult to leverage since the link between input values, parameter values and intended outcome is typically more opaque. To solve this problem, the method from Reference [44] synthesizes inputs with ground-truth labels by using systematic techniques to cover the modification space. Error tables are then created for all counterexamples, with table columns associated to input features (e.g., car model, environment or brightness for an image classifier used in autonomous driving). The analysis of this table can then provide a comprehensible explanation of failures, e.g., "The model does not identify white cars driving away from us on forest roads" [44]. These explanations support further data collection or augmentation in the next iteration of the ML workflow. In contrast, providing comprehensible results is much harder for formal verification methods that identify counterexamples based on proximity and smoothness; mapping such counterexamples to guide remedial actions these verification methods to remedial action remains an open challenge (MV07 in Table 6).

While adding context to training data helps inform how Data Management activities should be modified to improve model performance, no analogous methods exist for adjusting Model Learning activities (e.g., model and hyperparameter selection) in light of verification results. In general, defining a general method for performance improvement based on verification results is an open challenge (MV08 in Table 6).

The need for interpretable models is widely accepted; it is also an important part of verification evidence, being comprehensible to people not involved in the ML workflow. However, verifying that a model is interpretable is non-trivial, and a rigorous evaluation of interpretability is necessary. Doshi-Velez and Kim [42] suggest three possible approaches to achieving this verification: *Application grounded*, which involves placing the explanations into a real application and letting the end user test it; *Human grounded*, which uses lay humans rather than domain experts to test more general forms of explanation; and *Functionally grounded*, which uses formal definitions to evaluate the quality of explanations without human involvement.

6.5 Summary and Open Challenges

Table 5 summarises the assurance methods that can be applied during the Model Verification stage, listed in the order in which they were introduced earlier in this section. We note that the test-based verification methods outnumber the methods that use formal verification. Furthermore, the test-based methods are model agnostic, while the few formal verification methods that exist are largely restricted to neural networks and, due to their abstract nature, do little to support context or

Table 5. Assurance Methods for the Model Verification Stage

	Associated activities[†]			**Supported desiderata**[‡]		
	Requirement	Test-based	Formal	Compre-	Contextually	Compre-
Method	Encoding	Verification	Verification	hensive	Relevant	hensible
Independent derivation of test cases	✔	✓	✓		★	
Normal and robustness tests [143]	✓	✔		★		
Measure data coverage		✔		★	☆	
Measure model coverage [104, 126, 161]		✔		★	☆	
Guided fuzzing [123]		✔		★		
Combinatorial Testing [103]		✔		★		
SMT solvers [68]			✔	★		
Abstract Interpretation [58]			✔	★		
Generate tests via simulation		✔		★	☆	☆
Verifier of Random Forests [168]			✔	★		
Verification of ML Libraries [154]			✔	★		
Check for unwanted bias [16]		✔			★	
Use synthetic test data [167]	✔	✔		★	★	☆
Use GAN to inform test generation [190]		✔		★	★	
Incorporate system level semantics [45]	✔	✔		★	★	☆
Counterexample-guided data augmentation [44]		✔		★	☆	★
Probabilistic verification [173]			✔	★		
Use confidence levels [45]		✔	✓		☆	★
Evaluate interpretability [42]		✔	✔		★	★

[†] ✔ = activity that the method is typically used in; ✓ = activity that may use the method.
[‡] ★ = desideratum supported by the method; ☆ = desideratum partly supported by the method.

comprehension. Finally, the majority of the methods are concerned with the Comprehensive desideratum, while the Contextually Relevant and Comprehensible desiderata are poorly supported.

The open challenges for the assurance of the Model Verification stage are presented in Table 6. Much of the existing research for the testing and verification of ML models has focused on neural networks. Providing methods for other ML models remains an underexplored open challenge (MV03). A small number of typical errors have been identified but more work is required to develop methods for the detection and prevention of such errors (MV01). Measures of testing coverage are possible for ML models, however, understanding the benefits of a particular coverage remains an open challenge (MV02). Mapping model features to context presents challenges

Table 6. Open Challenges for the Assurance Concerns Associated with the Model Verification (MV) Stage

ID	Open Challenge	Desideratum (Section)
MV01	Understanding how to detect and protect against typical errors	Comprehensive (Section 6.4.1)
MV02	Test coverage measures with theoretical and empirical justification	
MV03	Formal verification for ML models other than neural networks	
MV04	Mapping requirements to model features	Contextually Relevant (Section 6.4.2)
MV05	General framework for synthetic test generation	
MV06	Mapping of model-free reinforcement learning states to real-world contexts	
MV07	Using proximity and smoothness violations to improve models	Comprehensible
MV08	General methods to inform training based on performance failures	(Section 6.4.3)

(MV04, MV06) both in the specification of requirements that maintain original intent and in the analysis of complex models. Furthermore, where context is incorporated into synthetic testing, this is achieved on a case by case basis and no general framework for such testing yet exists (MV05). Finally, although formal methods started to appear for the verification of ML models, they return counterexamples that are difficult to comprehend and cannot inform the actions that should be undertaken to improve model performance (MV07, MV08).

7 MODEL DEPLOYMENT

The aim of the ML workflow is to produce a model to be used as part of a system. The range of potential system types is vast. For example, it includes autonomous vehicles, personal healthcare applications, financial systems and deep space exploration. How the model is deployed within the system is a key consideration for an assurance argument. The last part of our survey focuses on the assurance of this deployment: We do not cover the complete assurance of the overall system, which represents a vast scope, well beyond what can be accomplished within this article. However, the close link between the model and the system means that this section inevitably includes some system-level considerations.

7.1 Inputs and Outputs

The key input artefacts to this stage of the ML lifecycle are a verified model and associated verification evidence. An understanding of the intended system use and an appreciation of the system architecture are also needed. The key output is the model, suitably deployed within a system.

7.2 Activities

7.2.1 Integration. This activity involves integrating the ML model into the wider system architecture. This requires linking system sensors to the model inputs. Likewise, model outputs need to be provided to the wider system. Shared or concurrent processing of data, alongside non-ML components, should be considered. A significant integration-related consideration is protecting the wider system against the effects of the occasional incorrect output from the ML model. This would be expected to be part of system-level safety analysis.

7.2.2 Monitoring. This activity considers the following types of monitoring associated with the deployment of an ML-developed model within a safety-critical system:

(1) Monitoring the *inputs* provided to the model. This could, for example, involve checking whether inputs are within acceptable bounds before they are provided to the ML model.

(2) Monitoring the *environment* in which the system is used. This type of monitoring can employ runtime verification techniques [53] and can be used, for example, to check that the observed environment matches any assumptions made during the ML workflow [9].
(3) Monitoring the *internals* of the model. This is useful, for example, to protect against the effects of single-event upsets, where environmental effects result in a change of state within a micro-electronic device [163]. Protection against single-event upsets may be dataset and ML approach dependent [175].
(4) Monitoring the *output* of the model. This replicates a traditional system safety approach in which a high-integrity monitor is used alongside a lower-integrity item.

7.2.3 Updating. Updates are an important part of any software intensive system [127]. Similarly, deployed ML models are expected to require updating during a system's life. This activity relates to managing and implementing these updates. Conceptually it also includes, as a special case, updates that occur as part of online learning (e.g., within the implementation of an RL-based model). However, since they are intimately linked to the model, these considerations are best addressed within the Model Learning stage.

7.3 Desiderata

From an assurance perspective, the deployed ML model should exhibit the following key properties:

(1) Fit-for-Purpose—This property recognises that the ML model needs to be fit for the intended purpose within the specific system context. In particular, it is possible for exactly the same model to be fit-for-purpose within one system, but not fit-for-purpose within another. Essentially, this property adopts a model-centric focus.
(2) Tolerable—This property acknowledges that it is typically unreasonable to expect ML models to achieve the same levels of reliability as traditional (hardware or software) components. Consequently, if ML models are to be used within safety-critical systems, then the wider system must be able to tolerate the occasional incorrect output from the ML model. Equivalently, from a system perspective, the ML model must be tolerable.
(3) Adaptable—This property is concerned with the ease with which modifications can be made to the deployed ML model. These modifications may be motivated by a variety of reasons, including changes in the operating, or legislative, environment. More generally, this property recognises the inevitability of change within a software system; consequently, it is closely linked to the updating activity described in Section 7.2.3.

7.4 Methods

This section considers each of the three desiderata in turn. Methods that can be applied during each Model Deployment activity, to help achieve the desired property, are discussed (Table 7).

7.4.1 Fit-for-Purpose. In order for an ML model to be fit-for-purpose within a given system deployment, there must be confidence that the performance observed during the Model Verification stage is representative of the deployed performance. This confidence could be negatively impacted by changes in computational hardware between the various stages of the ML lifecycle, e.g., different numerical representations can affect accuracy and energy use [66]. Issues associated with specialised hardware (e.g., custom processors designed for AI applications) may partly be addressed by suitable on-target testing (i.e., testing on the hardware used in the system deployment).

Differences between the inputs received during operational use and those provided during training and verification can result in levels of deployed performance that are very different to those observed during verification. There are several ways these differences can arise:

(1) Because the training (and verification) data were not sufficiently representative of the operational domain [38]. This could be a result of inadequate training data (specifically, the $\mathcal{O}$ subset referred to in Section 4), or it could be a natural consequence of a system being used in a wider domain than originally intended.
(2) As a consequence of failures in the subsystems that provide inputs to the deployed ML model (this relates to the $\mathcal{F}$ subset). Collecting, and responding appropriately to, health management information for relevant subsystems can help protect against this possibility.
(3) As a result of deliberate actions by an adversary (which relates to the $\mathcal{A}$ subset) [61, 162].
(4) Following changes in the underlying process to which the data are related. This could be a consequence of changes in the environment [4]. It could also be a consequence of changes in the way that people, or other systems, behave; this is especially pernicious when those changes have arisen as a result of people reacting to the model's behaviour.

The notion of the operational input distribution being different from that represented by the training data is referred to as distribution shift [118]. Most measures for detecting this rely on many operational inputs being available (e.g., Reference [179]). A box-based analysis of training data may allow detection on an input-by-input basis [12]. Learning-based approaches that compensate for distribution shift have been proposed [15, 18]. Nevertheless, especially for high-dimensional data [134], timely detection of distribution shift is an open challenge (MD01 in Table 8).

To demonstrate that a deployed model continues to remain fit-for-purpose, there needs to be a way of confirming that the model's internal behaviour is as designed. Equivalently, the provision of some form of **built-in test (BIT)** is helpful. A partial solution involves re-purposing traditional BIT techniques, including: watchdog timers [128], to provide confidence software is still executing, and behavioural monitors [85], to provide confidence software is behaving as expected (e.g., it is not claiming an excessive amount of system resources). However, these general techniques need to be supplemented by approaches specifically tailored for ML models [151].

For an ML model to be usable within a safety-critical system, it may be necessary for its output to be explainable (e.g., to support post-incident investigation). As discussed earlier, this is closely related to the Interpretable desideratum, discussed in Section 5. We also note that the open challenge relating to the global behaviour of a complex model (ML09 in Table 4) is relevant to the Model Deployment stage.

To support post-accident, or post-incident, investigations, sufficient information needs to be recorded to allow the ML model's behaviour to be subsequently explained. As a minimum, model inputs should be recorded; if the model's internal state is dynamic, then this should also be recorded. Furthermore, it is very likely that accident, or incident, investigation data will have to be recorded on a continual basis, and in such a way that it will usable after a crash and is protected against inadvertent (or deliberate) alteration. Understanding what information needs to be recorded, at what frequency and for how long it needs to be maintained is an open challenge (MD02 in Table 8).

7.4.2 Tolerable. To tolerate occasional incorrect outputs from a deployed ML model, the system needs to do two things. Firstly, it needs to detect when an output is incorrect. Secondly, it needs to replace the incorrect output with a suitable value to allow system processing activities to continue.

An ML model may produce an incorrect output when used outside the intended operational environment. This could be detected by monitoring for distribution shift, as indicated in the preceding section, possibly alongside monitoring the environment. An incorrect output may also be produced if the model is provided with inappropriate inputs. Again, this could be detected by monitoring for distribution shift or by monitoring the health of the system components that provide inputs to the model. In critical applications, it may be appropriate to define a minimum equipment list. This list should describe the equipment that must be present and functioning correctly to allow safe use of the system [119]. This approach can also protect the deployed ML model against system-level changes that would inadvertently affect its performance.

It may also be possible for the ML model to calculate its own "measure of confidence," which could be used to support the detection of an incorrect output. The ICM has been proposed [172], but this requires a distance metric to be defined on the input domain, which can be difficult. More generally, deriving an appropriate measure of confidence is an open challenge (MD03 in Table 8).

Another way of detecting incorrect outputs involves comparing them with "reasonable" values. This could, for example, take the form of introducing a simple monitor, acting directly on the output provided by the ML model [23]. If the monitor detects an invalid output, then the model is re-run (with the same input, if the model is non-deterministic, or with a different input). Defining a monitor that protects safety is possible [105], but providing sufficient protection yet still allowing the ML model sufficient freedom in behaviour, so that the benefits of using an ML-based approach can be realised, is an open challenge (MD04 in Table 8).

The difficulty with defining a monitor may be overcome by using multiple, 'independent' ML models, along with an "aggregator" that combines their multiple outputs into a single output. This can be viewed as an ML-based implementation of the concept of n-version programming [34]. The approach has some similarity to ensemble learning [147], but its motivation is different: ensemble learning aims to improve performance in a general sense, while using multiple, independent models as part of a system architecture aims to protect against the consequences of a single model occasionally producing an incorrect output. Whilst this approach may have value, it is not clear how much independence can be achieved, especially if models are trained from data that have a common generative model [49]. Consequently, understanding the level of independence that can be introduced into models trained on the same data is an open challenge (MD05 in Table 8).

If an incorrect output is detected, then, as noted above, a replacement value needs to be provided to the rest of the system. Runtime verification methods have been introduced in which this is achieved using software components developed and verified using traditional techniques [32, 65, 78]. In the Simplex architecture [78], a verified supervisory controller switches control from an unverifiable 'smart' controller to a verified safety controller when the former misbehaves. Alternatively, a fixed "safe" value or the "last-good" output provided by the ML model could be used. In this approach, a safety switch monitors the output from the ML model. If this is invalid, then the switch is thrown and the output from the 'alternative' approach is used instead. This assumes that an invalid output can be detected and, furthermore, that a substitute, potentially suboptimal, output can be provided in such cases.

The monitor, aggregator and switch model-deployment architectures could readily accommodate human interaction. For example, a human could play the role of a monitor, or that allocated to traditional software (e.g., when an autonomous vehicle hands control back to a human driver).

7.4.3 Adaptable. Like all software, a deployed ML model would be expected to change during the lifetime of the system in which it is deployed [127]. Indeed, the nature of ML, especially the possibility of operational systems capturing data that can be used to train subsequent versions of a model, suggests that ML models may change more rapidly than is the case for traditional software.

A key consideration when allowing ML models to be updated is the management of the change from an old version of a model to a new version. Several approaches can be used for this purpose:

(1) Placing the system in a "safe state" for the duration of the update process. In the case of an autonomous vehicle, this state could be stationary, with the parking brake applied, with no occupants and with all doors locked. In addition, updates could be restricted to certain geographic locations (e.g., the owner's driveway or the supplier's service area). The safe state could be determined by a technical process, or it could be specified by the user [127].
(2) If it is not feasible, or desirable, for the system to be put into a safe state, then an alternative is for the system to run two identical channels, one of which is "live" and the other of which is a "backup." The live model can be used whilst the backup is updated. Once the update is complete, the backup can become live and the other channel can be updated. This is a specific implementation of the A/B swap discussed in Reference [27].
(3) Another alternative is to use an approach deliberately designed to enable runtime code replacement (or "hot code loading"). This functionality is available, e.g., within Erlang [31].

ML model updating resembles the use of field-loadable software in the aerospace domain [143]. As such, several considerations are common to both activities: detecting corrupted or partially loaded software; checking compatibility; and preventing inadvertent triggering of the loading function. Approaches for protecting against corrupted updates should cover inadvertent data changes and deliberate attacks aiming to circumvent this protection [112].

In the common scenario where multiple instances of the same system have be deployed (e.g., when a manufacturer sells many units of an autonomous vehicle or medical diagnosis system) updates need to be managed at the "fleet" level. There may, for example, be a desire to gradually roll out an update so that its effects can be measured, taking due consideration of any ethical issues associated with such an approach. More generally, there is a need to monitor and control fleet-wide diversity [14]. Understanding how best to do this is an open challenge (MD06 in Table 8).

7.5 Summary and Open Challenges

Table 7 summarises assurance methods associated with the Model Deployment stage, matched to the associated activities and desiderata. The table shows that only two methods support the activity of updating an ML model. This may reflect the current state of the market for autonomous systems: There are few, if any, cases where a manufacturer has a large number of such systems in operational use. Given the link between the updating activity and the Adaptable desideratum, similar reasons may explain the lack of methods to support an adaptable system deployment.

The open challenges associated with the System Deployment stage (Table 8) include the following: concerns that extend to the Model Learning and Model Verification stages, e.g., providing measures of confidence (MD03); concerns that relate to system architectures, e.g., detecting distribution shift (MD01), supporting incident investigations (MD02), providing suitably flexible monitors (MD04) and understanding independence (MD05); and concerns that apply to system "fleets" (MD06).

8 CONCLUSION

Recent advances in machine learning underpin the development of many successful systems. ML technology is increasingly at the core of sophisticated functionality provided by smart devices, household appliances and online services, often unbeknownst to their users. Despite the diversity of these ML applications, they share a common characteristic: none is safety critical. Extending the success of machine learning to safety-critical systems holds great potential for application

Table 7. Assurance Methods for the Model Deployment Stage

	Associated activities[†]			**Supported desiderata**[‡]		
Method	Integration	Monitoring	Updating	Fit-for-Purpose	Tolerable	Adaptable
Use the same numerical precision for training and operation	✔			★		
Establish WCET [184]	✔			★	☆	
Monitor for distribution shift [118], [12]	✓	✔		★	★	
Implement general BIT [128], [85], [151]	✔	✔		★	★	
Explain an individual output [138]	✓	✔		★		
Record information for post-accident (or post-incident) investigation	✔			★		
Monitor the environment [9]		✔		★	★	
Monitor health of input-providing subsystems		✔		★	★	
Provide a confidence measure [172]	✔	✔			★	
Use an architecture that tolerates incorrect outputs [23], [32], [34]		✔			★	
Manage the update process [143]		✓	✔			★
Control fleet-wide diversity [14]			✔			★

[†] ✔ = activity that the method is typically used in; ✓ = activity that may use the method.
[‡] ★ = desideratum supported by the method; ☆ = desideratum partly supported by the method.

Table 8. Open Challenges for the Assurance Concerns Associated with the Model Deployment (MD) Stage

ID	**Open Challenge**	**Desideratum (Section)**
MD01	Timely detection of distribution shift, especially for high-dimensional datasets	Fit-for-Purpose (Section 7.4.1)
MD02	Information recording to support accident or incident investigation	
MD03	Providing a suitable measure of confidence in ML model output	Tolerable (Section 7.4.2)
MD04	Defining suitably flexible safety monitors	
MD05	Understanding the level of independence that can be introduced into models trained on the same data	
MD06	Monitoring and controlling fleet-wide diversity	Adaptable (Section 7.4.3)

domains ranging from healthcare and transportation to manufacturing, but requires the assurance of the ML models deployed within such systems. Our article explained that this assurance must cover all stages of the ML lifecycle, defined assurance desiderata for each such stage, surveyed the methods available to achieve these desiderata, and highlighted remaining open challenges.

For the Data Management stage, our survey shows that a wide range of data collection, preprocessing, augmentation and analysis methods can help ensure that ML training and verification datasets are Relevant, Complete, Balanced, and Accurate. Nevertheless, further research is required to devise methods capable of demonstrating that these data are sufficiently secure, fit-for-purpose and, when simulation is used to synthesise data, that simulations are suitably realistic.

The Model Learning stage has been the focus of tremendous research effort, and a vast array of model selection and learning methods are available to support the development of Performant and Robust ML models. In contrast, there is a significant need for additional hyperparameter selection and transfer learning methods, and for research into ensuring that ML models are Reusable and Interpretable, in particular through providing context-relevant explanations of behaviour.

Assurance concerns associated with the Model Verification stage are addressed by numerous test-based verification methods and by a small repertoire of recently introduced formal verification methods. The verification results provided by these methods are often Comprehensive (for the ML model aspects being verified) and, in some cases, Contextually Relevant. However, there are currently insufficient methods capable of encoding the requirements of the model being verified into suitable tests and formally verifiable properties. Furthermore, ensuring that verification results are Comprehensible is still very challenging.

The integration and monitoring activities from the Model Deployment stage are supported by a sizeable set of methods that can help address the Fit-for-Purpose and Tolerable desiderata of deployed ML models. These methods are often inspired by analogous methods for the integration and monitoring of software components developed using traditional engineering approaches. In contrast, ML model updating using data collected during operation has no clear software engineering counterpart. As such, model updating methods are scarce and typically unable to provide the assurance needed to deploy ML models that are Adaptable within safety-critical systems.

A further open challenge for assuring the ML lifecycle is the determination of effective tradeoffs between the desiderata over the four stages. In many applications, the desiderata are closely related, either reinforcing each other or placing competing demands on the ML lifecycle activities. As an example, a highly Reusable ML model will likely be easily Adaptable, but might be less Contextually Relevant. Further tradeoffs arise in the assurance domain, e.g., an Explainable model might introduce vulnerabilities exploitable in a cyberattack [77].

This brief summary shows that considerable research is still needed to address outstanding assurance concerns associated with every stage of the ML lifecycle. In general, using ML components within safety-critical systems poses numerous open challenges. At the same time, the research required to address these challenges can build on a promising combination of rigorous methods developed by several decades of sustained advances in machine learning, in software and systems engineering, and in assurance development.

REFERENCES

[1] Mahdieh Abbasi, Arezoo Rajabi, Azadeh Sadat Mozafari, Rakesh B. Bobba, and Christian Gagne. 2018. Controlling over-generalization and its effect on adversarial examples generation and detection. arXiv:1808.08282. Retrieved from https://arxiv.org/abs/1808.08282.

[2] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.

[3] Ajaya Adhikari, D. M. Tax, Riccardo Satta, and Matthias Fath. 2018. Example and Feature importance-based Explanations for Black-box Machine Learning Models. arXiv:1812.09044. Retrieved from https://arxiv.org/abs/1812.09044.

[4] Rocío Alaiz-Rodríguez and Nathalie Japkowicz. 2008. Assessing the impact of changing environments on classifier performance. In *Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence*. Springer, 13–24.

[5] Rob Alexander, Heather Rebecca Hawkins, and Andrew John Rae. 2015. *Situation Coverage—A Coverage Criterion for Testing Autonomous Robots*. Technical Report YCS-2015-496. Department of Computer Science, University of York.

[6] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. 2018. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *Int. J. Comput. Vis.* 126, 9 (2018), 961–972.

[7] Maksym Andriushchenko and Matthias Hein. 2019. Provably robust boosted decision stumps and trees against adversarial attacks. In *Advances in Neural Information Processing Systems*. 13017–13028.

[8] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. 2012. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the International Workshop on Ambient Assisted Living*. 216–223.

[9] Adina Aniculaesei, Daniel Arnsberger, Falk Howar, and Andreas Rausch. 2016. Towards the verification of safety-critical autonomous systems in dynamic environments. In *Proceedings of the Workshop on Verification and Validation of Cyber-Physical Systems (V2CPS@IFM'16)*. 79–90.

[10] Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2017. Data augmentation generative adversarial networks. arXiv:1711.04340. Retrieved from https://arxiv.org/abs/1711.04340.

[11] Maziar Arjomandi, Shane Agostino, Matthew Mammone, Matthieu Nelson, and Tong Zhou. 2006. *Classification of Unmanned Aerial Vehicles. Report for Mechanical Engineering Class*. Technical Report. University of Adelaide, Australia.

[12] Rob Ashmore and Matthew Hill. 2018. Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift. In *Proceedings of the International Conference on Computer Safety, Reliability, and Security*. Springer, 393–405.

[13] Rob Ashmore and Elizabeth Lennon. 2017. Progress towards the assurance of non-traditional software. In *Developments in System Safety Engineering, Proceedings of the 25th Safety-Critical Systems Symposium*. 33–48.

[14] Rob Ashmore and Bhopinder Madahar. 2019. Rethinking diversity in the context of autonomous systems. In *Engineering Safe Autonomy, Proceedings of the 27th Safety-Critical Systems Symposium*. 175–192.

[15] Kamyar Azizzadenesheli, Anqi Liu, Fanny Yang, and Animashree Anandkumar. 2019. Regularized learning for domain adaptation under label shifts. arXiv:1903.09734. Retrieved from https://arxiv.org/abs/1903.09734.

[16] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilović, S. Nagar, K. N. Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. 2019. AI fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM J. Res. Dev.* 63, 4/5 (2019), 4:1–4:15.

[17] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13(Feb.2012), 281–305.

[18] Steffen Bickel, Michael Brückner, and Tobias Scheffer. 2009. Discriminative learning under covariate shift. *J. Mach. Learn. Res.* 10, 9 (2009), 2137–2155.

[19] Arijit Bishnu, Sameer Desai, Arijit Ghosh, Mayank Goswami, and Paul Subhabrata. 2015. Uniformity of point samples in metric spaces using gap ratio. In *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation*. 347–358.

[20] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

[21] Robin Bloomfield and Peter Bishop. 2010. Safety and assurance cases: Past, present and possible future—An Adelard perspective. In *Making Systems Safer*. Springer, 51–67.

[22] Barry Boehm and Wilfred J. Hansen. 2000. *Spiral Development: Experience, Principles, and Refinements*. Technical Report CMU/SEI-2000-SR-008. Carnegie Mellon University.

[23] Chris Bogdiukiewicz, Michael Butler, Thai Son Hoang, Martin Paxton, James Snook, Xanthippe Waldron, and Toby Wilkinson. 2017. Formal development of policing functions for intelligent systems. In *Proceedings of the 28th International Symposium on Software Reliability Engineering*. IEEE, 194–204.

[24] Andrew P. Bradley. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recogn.* 30, 7 (1997), 1145–1159.

[25] Houssem Ben Braiek and Foutse Khomh. 2018. On Testing Machine Learning Programs. arXiv:1812.02257. Retrieved from https://arxiv.org/abs/1812.02257.

[26] Carla E. Brodley and Mark A. Friedl. 1999. Identifying mislabeled training data. *J. Artif. Intell. Res.* 11 (1999), 131–167.

[27] Atilla Bulmus, Axel Freiwald, and Chris Wunderlich. 2017. *Over the Air Software Update Realization within Generic Modules with Microcontrollers Using External Serial FLASH*. Technical Report. SAE Technical Paper.

[28] Jonathod Byrd and Zachary Lipton. 2019. What is the effect of importance weighting in deep learning? arXiv:1812.03372. Retrieved from https://arxiv.org/abs/1812.03372.

[29] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2018. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Trans. Softw. Eng.* 44, 11 (2018), 1039–1069.

[30] Cristian S. Calude and Giuseppe Longo. 2017. The deluge of spurious correlations in big data. *Found. Sci.* 22, 3 (2017), 595–612.

[31] Richard Carlsson, Björn Gustavsson, Erik Johansson, Thomas Lindgren, Sven-Olof Nyström, Mikael Pettersson, and Robert Virding. 2000. *Core Erlang 1.0 Language Specification.* Technical Report. Information Technology Department, Uppsala University.

[32] Paul Caseley. 2016. Claims and architectures to rationate on automatic and autonomous functions. In *Proceedings of the 11th International Conference on System Safety and Cyber-Security.* IET, 1–6.

[33] Nitesh V. Chawla, Aleksandar Lazarevic, Lawrence O. Hall, and Kevin W. Bowyer. 2003. SMOTEBoost: Improving prediction of the minority class in boosting. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery.* 107–119.

[34] Liming Chen and Algirdas Avizienis. 1978. N-version programming: A fault-tolerance approach to reliability of software operation. In *Proceedings of the 8th IEEE International Symposium on Fault-Tolerant Computing*, Vol. 1. 3–9.

[35] Xinyun Chen, Chang Liu, Bo Li, Kimberley Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. arXiv:1712.05526. Retrieved from https://arxiv.org/abs/1712.05526.

[36] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Zakaria Anil, Rohan an Haque, Lichan Hong, Vihan Jain, Xiabing Liu, and Hemal Shah. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems.* ACM, 7–10.

[37] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2018. Back to basics: Benchmarking canonical evolution strategies for playing Atari. arXiv:1802.08842. Retrieved from https://arxiv.org/abs/1802.08842.

[38] David A. Cieslak and Nitesh V. Chawla. 2009. A framework for monitoring classifiers performance: When and why failure occurs? *Knowl. Inf. Syst.* 18, 1 (2009), 83–108.

[39] Adnan Darwiche. 2018. Human-level intelligence or animal-like abilities? *Comm. ACM* 61, 10 (2018), 56–67.

[40] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 248–255.

[41] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2017. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 3 (2017), 653–664.

[42] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. arXiv:1702.08608. Retrieved from https://arxiv.org/abs/1702.08608.

[43] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. 2019. VERIFAI: A toolkit for the design and analysis of artificial intelligence-based systems. arXiv:1902.04245. Retrieved from https://arxiv.org/abs/1902.04245.

[44] Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Kurt Keutzer, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2018. Counterexample-guided data augmentation. arXiv:1805.06962. Retrieved from https://arxiv.org/abs/1805.06962.

[45] Tommaso Dreossi, Somesh Jha, and Sanjit A. Seshia. 2018. Semantic adversarial deep learning. arXiv:1804.07045. Retrieved from https://arxiv.org/abs/1804.07045.

[46] Chris Drummond and Robert C. Holte. 2006. Cost curves: An improved method for visualizing classifier performance. *Mach. Learn.* 65, 1 (2006), 95–130.

[47] Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2019. Sherlock-A tool for verification of neural network feedback systems: Demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control.* 262–263.

[48] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis.* Springer, 269–286.

[49] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. 2018. Adversarial vulnerability for any classifier. arXiv:1802.08686. Retrieved from https://arxiv.org/abs/1802.08686.

[50] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2015. Fundamental limits on adversarial robustness. In *Proceedings of the ICML Workshop on Deep Learning.*

[51] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. 2016. Robustness of classifiers: From adversarial to random noise. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (NIPS'16). Curran Associates Inc., Red Hook, NY, 1632–1640.

[52] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 259–268.

[53] Angelo Ferrando, Louise A. Dennis, Davide Ancona, Michael Fisher, and Viviana Mascardi. 2018. Verifying and validating autonomous systems: Towards an integrated approach. In *Proceedings of the International Conference on Runtime Verification.* Springer, 263–281.

[54] Peter Flach. 2019. Performance evaluation in machine learning: The good, the bad, the ugly and the way forward. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*. 9808–9814.

[55] Michael Forsting. 2017. Machine learning will change medicine. *J. Nucl. Med.* 58, 3 (2017), 357–358.

[56] Yoav Freund, Robert Schapire, and Naoki Abe. 1999. A short introduction to boosting. *J. Jpn. Soc. Artif. Intell.* 14, 771–780 (1999), 1612.

[57] Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* 55, 1 (1997), 119–139.

[58] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. IEEE, 3–18.

[59] Aurélien Géron. 2017. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc.

[60] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. Vol. 1. MIT Press.

[61] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv:1412.6572. Retrieved from https://arxiv.org/abs/1412.6572.

[62] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying vulnerabilities in the machine learning model supply chain. arXiv:1708.06733. Retrieved from https://arxiv.org/abs/1708.06733.

[63] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. 2017. Learning from class-imbalanced data: Review of methods and applications. *Expert Syst. Appl.* 73 (2017), 220–239.

[64] Jeff Heaton. 2016. An empirical analysis of feature engineering for predictive modeling. In *Proceedings of SoutheastCon'16*. IEEE, 1–6.

[65] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. 1996. Automated consistency checking of requirements specifications. *ACM Trans. Softw. Eng. Methodol.* 5, 3 (1996), 231–261.

[66] Parker Hill, Babak Zamirai, Shengshuo Lu, Yu-Wei Chao, Michael Laurenzano, Mehrzad Samadi, Marios C. Papaefthymiou, Scott A. Mahlke, Thomas F. Wenisch, Jia Deng, Lingjia Tang, and Jason Mars. [n.d.]. Rethinking Numerical Representations for Deep Neural Networks. arXiv:1808.02513. Retrieved from https://arxiv.org/abs/1808.02513.

[67] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. [n.d.]. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580. Retrieved from https://arxiv.org/abs/1207.0580.

[68] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification*, Rupak Majumdar and Viktor Kuncak (Eds.), Lecture Notes in Computer Science, Vol. 10426. Springer, 3–29.

[69] Zhongling Huang, Zongxu Pan, and Bin Lei. 2017. Transfer learning with deep convolutional neural network for SAR target classification with limited labeled data. *Remote Sens.* 9, 9 (2017), 907.

[70] Casidhe Hutchison, Milda Zizyte, Patrick E. Lanigan, David Guttendorf, Michael Wagner, Claire Le Goues, and Philip Koopman. 2018. Robustness testing of autonomy software. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice*. 276–285.

[71] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. 2015. Beyond manual tuning of hyperparameters. *Künstl. Intell.* 29, 4 (2015), 329–337.

[72] Didac Gil De La Iglesia and Danny Weyns. 2015. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Trans. Auton. Adapt. Syst.* 10, 3 (2015), 15.

[73] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167. Retrieved from https://arxiv.org/abs/1502.03167.

[74] Bandar Seri Iskandar. 2017. Terrorism detection based on sentiment analysis using machine learning. *J. Eng. Appl. Sci.* 12, 3 (2017), 691–698.

[75] ISO. 2018. *Road Vehicles—Functional Safety: Part 6*. Technical Report BS ISO 26262-6:2018. ISO.

[76] Nathalie Japkowicz. 2001. Concept-learning in the presence of between-class and within-class imbalances. In *Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence*. Springer, 67–77.

[77] Nikita Johnson and Tim Kelly. 2019. Devil's in the detail: Through-life safety and security co-assurance using SSAF. In *Proceedings of the 38th International Conference on Computer Safety, Reliability, and Security*. Springer, 299–314.

[78] Taylor T. Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. 2016. Real-time reachability for verified Simplex design. *ACM Trans. Embed. Comput. Syst.* 15, 2 (2016), 1–27.

[79] M. H. Kabir, M. R. Hoque, H. Seo, and S. H. Yang. 2015. Machine learning based adaptive context-aware system for smart home environment. *Int. J. Smart Home* 9, 11 (2015), 55–62.

[80] Faisal Kamiran and Toon Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowl. Inf. Syst.* 33, 1 (2012), 1–33.

[81] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proceedings of the International Conference on Computer Aided Verification*. Springer, 97–117.

[82] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *Proceedings of the International Conference on Computer Aided Verification*. Springer, 443–452.

[83] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. 2012. Leakage in data mining: Formulation, detection, and avoidance. *ACM Trans. Knowl. Discov. Data* 6, 4 (2012), 15.

[84] Jeffrey O. Kephart and David M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[85] Muhammad Taimoor Khan, Dimitrios Serpanos, and Howard Shrobe. 2016. A rigorous and efficient run-time security monitor for real-time critical embedded system applications. In *Proceedings of the 3rd World Forum on Internet of Things*. IEEE, 100–105.

[86] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2018. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 3407–3414.

[87] Roger E. Kirk. 2007. Experimental design. Wiley Online Library.

[88] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. 2015. Audio augmentation for speech recognition. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association*.

[89] Patrick Koch, Brett Wujek, Oleg Golovidov, and Steven Gardner. 2017. Automated hyperparameter tuning for effective machine learning. In *Proceedings of the SAS Global Forum Conference*.

[90] Matthieu Komorowski, Leo A. Celi, Omar Badawi, Anthony C. Gordon, and A. Aldo Faisal. 2018. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nat. Med.* 24, 11 (2018), 1716–1720.

[91] Philip Koopman and Frank Fratrik. 2019. How many operational design domains, objects, and events? In *Proceedings of the AAAI Workshop on Artificial Intelligence Safety*.

[92] Philip Koopman, Aaron Kane, and Jen Black. 2019. Credible autonomy safety argumentation. In *Proceedings of the 27th Safety-Critical Systems Symposium*.

[93] S. B. Kotsiantis, Dimitris Kanellopoulos, and P. E. Pintelas. 2006. Data preprocessing for supervised leaning. *Int. J. Comput. Sci.* 1, 2 (2006), 111–117.

[94] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas. 2007. Data preprocessing for supervised leaning. *Int. J. Comput. Electr. Autom. Contr. Inf. Eng.* 1, 12 (2007), 4104–4109.

[95] Samantha Krening, Brent Harrison, Karen M. Feigh, Charles Lee Isbell, Mark Riedl, and Andrea Thomaz. 2017. Learning from explanations using sentiment and advice in RL. *IEEE Trans. Cogn. Dev. Syst.* 9, 1 (2017), 44–55.

[96] Isaac Lage, Andrew Ross, Kim Been, Samuel Gershman, and Finale Doshi-Velez. 2018. Human-in-the-loop interpretability prior. In *Proceedings of the Conference on Neural Information Processing Systems*. 10180–10189.

[97] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2323.

[98] Joseph Lemley, Filip Jagodzinski, and Razvan Andonie. 2016. Big holes in big data: A Monte Carlo algorithm for detecting large hyper-rectangles in high dimensional data. In *Proceedings of the IEEE Computer Software and Applications Conference*. 563–571.

[99] Zachary C. Lipton. 2016. The mythos of model interpretability. arXiv:1606.03490. Retrieved from https://arxiv.org/abs/1606.03490.

[100] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1265–1282.

[101] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. 2013. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Inf. Sci.* 250 (2013), 113–141.

[102] Gustavo A. Lujan-Moreno, Phillip R. Howard, Omar G. Rojas, and Douglas C. Montgomery. 2018. Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study. *Expert Syst. Appl.* 109 (2018), 195–205.

[103] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic combinatorial testing for deep learning systems. In *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 614–618.

[104] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 120–131.

[105] Mathilde Machin, Jérémie Guiochet, Hélène Waeselynck, Jean-Paul Blanquart, Matthieu Roy, and Lola Masson. 2018. SMOF: A safety monitoring framework for autonomous systems. *IEEE Trans. Syst. Man Cybernet. Syst.* 48, 5 (2018), 702–715.

[106] Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 5188–5196.

[107] Spyros Makridakis. 2017. The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms. *Futures* 90 (2017), 46–60.

[108] Pedro Marcelino. 2018. Transfer learning from pre-trained models. In *Towards Data Science* (2018).

[109] George Mason, Radu Calinescu, Daniel Kudenko, and Alec Banks. 2017. Assured reinforcement learning with formally verified abstract policies. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence.* 105–117.

[110] Michael Maurer, Ivan Breskovic, Vincent C. Emeakaroha, and Ivona Brandic. 2011. Revealing the MAPE loop for the autonomic management of cloud infrastructures. In *Proceedings of the Symposium on Computers and Communications.* IEEE, 147–152.

[111] Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner. 2016. *Autonomous Driving: Technical, Legal and Social Aspects.* Springer Nature.

[112] Christopher Meyer and Jörg Schwenk. 2013. SoK: Lessons learned from SSL/TLS attacks. In *Proceedings of the International Workshop on Information Security Applications.* Springer, 189–209.

[113] Microsoft. 2019. How to choose algorithms for Azure Machine Learning Studio. Retrieved February 2019 from https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice.

[114] Tom M. Mitchell. 1997. *Machine Learning.* McGraw–Hill.

[115] Model Zoos Caffe 2019. Caffe Model Zoo. Retrieved March 2019 from http://caffe.berkeleyvision.org/model_zoo.html.

[116] Model Zoos Github 2019. Model Zoos of machine and deep learning technologies. Retrieved March 2019 from https://github.com/collections/ai-model-zoos.

[117] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1765–1773.

[118] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recogn.* 45, 1 (2012), 521–530.

[119] Pamela A. Munro and Barbara G. Kanki. 2003. An analysis of ASRS maintenance reports on the use of minimum equipment lists. In *Proceedings of the 12th International Symposium on Aviation Psychology.*

[120] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective.* The MIT Press.

[121] Partha Niyogi and Federico Girosi. 1996. On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. *Neural Comput.* 8, 4 (1996), 819–842.

[122] Object Management Group. 2018. Structured Assurance Case Metamodel (SACM). Version 2.0.

[123] Augustus Odena and Ian Goodfellow. 2018. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. arXiv:1807.10875. Retrived from https://arxiv.org/abs/1807.10875.

[124] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1717–1724.

[125] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security.* ACM, 506–519.

[126] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles.* ACM, 1–18.

[127] Teresa Placho, Christoph Schmittner, Arndt Bonitz, and Oliver Wana. 2020. Management of automotive software updates. *Microprocess. Microsystems.* 78 (2020), 103257.

[128] Michael J. Pont and Royan H. L. Ong. 2002. Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study. In *Proceedings of the 1st Nordic Conference on Pattern Languages of Programs.*

[129] Lutz Prechelt. 1998. Early stopping-but when? In *Neural Networks: Tricks of the Trade.* Springer, 55–69.

[130] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. 2018. Tunability: Importance of hyperparameters of machine learning algorithms. arXiv:1802.09596. Retrieved from https://arxiv.org/abs/1802.09596.

[131] Foster Provost and Tom Fawcett. 2001. Robust classification for imprecise environments. *Mach. Learn.* 42, 3 (2001), 203–231.

[132] J. Provost Foster, Fawcett Tom, and Kohavi Ron. 1998. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning.* 445–453.

[133] R-Bloggers Data Analysis 2019. How to Use Data Analysis for Machine Learning. Retrieved February 2019 from https://www.r-bloggers.com/how-to-use-data-analysis-for-machine-learning-example-part-1.

[134] Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. 2019. Failing loudly: An empirical study of methods for detecting dataset shift. *Advances in Neural Information Processing Systems* 32 (2019).

[135] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. 2007. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proceedings of the European Conference on Machine Learning*. Springer, 699–707.

[136] Francesco Ranzato and Marco Zanella. 2019. Robustness verification of support vector machines. In *Proceedings of the International Static Analysis Symposium*. Springer, 271–295.

[137] Jorge-L. Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. 2016. Transition-aware human activity recognition using smartphones. *Neurocomputing* 171 (2016), 754–767.

[138] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1135–1144.

[139] F. Ricci, L. Rokach, and B. Shapira. 2015. Recommender systems: Introduction and challenges. *Recommender Systems Handbook* (2015), 1–34.

[140] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. 2016. The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3234–3243.

[141] Andrew Slavin Ross and Finale Doshi-Velez. 2018. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 1660–1669.

[142] Saharon Rosset, Claudia Perlich, Grzergorz Świrszcz, Prem Melville, and Yan Liu. 2010. Medical data mining: Insights from winning two competitions. *Data Min. Knowl. Discov.* 20, 3 (2010), 439–468.

[143] RTCA. 2011. *Software Considerations in Airborne Systems and Equipment Certification*. Technical Report DO-178C.

[144] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1, 5 (2019), 206–215.

[145] Stuart J. Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited.

[146] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. 1989. Design and analysis of computer experiments. *Stat. Sci.* (1989), 409–423.

[147] Omer Sagi and Lior Rokach. 2018. Ensemble learning: A survey. *Data Min. Knowl. Discov.* 8, 4 (2018), e1249.

[148] Ahmed Salem, Michael Backes, and Yang Zhang. 2020. Don't Trigger Me! A Triggerless Backdoor Attack Against Deep Neural Networks. arXiv:2010.03282. Retrieved from https://arxiv.org/abs/2010.03282.

[149] Robert G. Sargent. 2009. Verification and validation of simulation models. In *Proceedings of the Winter Simulation Conference*. 162–176.

[150] Lawrence K. Saul and Sam T. Roweis. 2003. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *J. Mach. Learn. Res.* 4(Jun.2003), 119–155.

[151] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Efficient on-line error detection and mitigation for deep neural network accelerators. In *Proceedings of the International Conference on Computer Safety, Reliability, and Security*. Springer, 205–219.

[152] Scikit-Taxonomy 2019. Scikit—Choosing the right estimator. Retrieved February 2019 from https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.

[153] Noam Segev, Maayan Harel, Shie Mannor, Koby Crammer, and Ran El-Yaniv. 2017. Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 9 (2017), 1811–1824.

[154] Daniel Selsam, Percy Liang, and David L. Dill. 2017. Developing bug-free machine learning systems with formal mathematics. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3047–3056.

[155] Victor S. Sheng and Jing Zhang. 2019. Machine learning with crowdsourcing: A brief summary of the past research and future directions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 9837–9843.

[156] Andy Shih, Arthur Choi, and Adnan Darwiche. 2018. Formal verification of Bayesian network classifiers. In *Proceedings of the International Conference on Probabilistic Graphical Models*. 427–438.

[157] Padhraic Smyth. 1996. Bounds on the mean classification error rate of multiple experts. *Pattern Recogn. Lett.* 17, 12 (1996), 1253–1257.

[158] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage.* 45, 4 (2009), 427–437.

[159] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[160] Sanatan Sukhija, Narayanan C. Krishnan, and Deepak Kumar. 2018. Supervised heterogeneous transfer learning using random forests. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*. ACM, 157–166.

[161] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 109–119.

[162] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. arXiv:1312.6199. Retrieved from https://arxiv.org/abs/1312.6199.

[163] A. Taber and E. Normand. 1993. Single event upset in avionics. *IEEE Trans. Nucl. Sci.* 40, 2 (1993), 120–126.

[164] Mariarosaria Taddeo, Tom McCutcheon, and Luciano Floridi. 2019. Trusting artificial intelligence in cybersecurity is a double-edged sword. *Nat. Mach. Intell.* (2019), 557–560.

[165] Luke Taylor and Geoff Nitschke. 2017. Improving deep learning using generic data augmentation. arXiv:1708.06020. Retrieved from https://arxiv.org/abs/1708.06020.

[166] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 847–855.

[167] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, 303–314.

[168] John Törnblom and Simin Nadjm-Tehrani. 2018. Formal verification of random forests in safety-critical applications. In *Proceedings of the International Workshop on Formal Techniques for Safety-Critical Systems*. Springer, 55–71.

[169] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of deep convolutional neural networks using ImageStars. arXiv:2004.05511. Retrieved from https://arxiv.org/abs/2004.05511.

[170] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. arXiv:2004.05519. Retrieved from https://arxiv.org/abs/2004.05519.

[171] John W. Tukey. 1977. *Exploratory Data Analysis*. Vol. 2. Reading, MA.

[172] Jasper van der Waa, Jurriaan van Diggelen, Mark A Neerincx, and Stephan Raaijmakers. 2018. ICM: An intuitive model independent and accurate certainty measure for machine learning.. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART'18)*. 314–321.

[173] Perry Van Wesel and Alwyn E. Goodloe. 2017. Challenges in the verification of reinforcement learning algorithms. (2017).

[174] Kiri Wagstaff. 2012. Machine learning that matters. arXiv:1206.4656. Retrieved from https://arxiv.org/abs/1206.4656.

[175] Kiri L. Wagstaff and Benjamin Bornstein. 2009. K-means in space: A radiation sensitivity evaluation. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1097–1104.

[176] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *Proceedings of the International Conference on Machine Learning*. 1058–1066.

[177] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing hyperparameters in machine learning. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. IEEE, 36–52.

[178] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*. IEEE, 707–723.

[179] Ke Wang, Senqiang Zhou, Chee Ada Fu, and Jeffrey Xu Yu. 2003. Mining changes of classification by correspondence tracing. In *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM, 95–106.

[180] Lu Wang, Xuanqing Liu, Jinfeng Yi, Zhi-Hua Zhou, and Cho-Jui Hsieh. 2019. Evaluating the robustness of nearest neighbor classifiers: A primal-dual perspective. arXiv:1906.03972. Retrieved from https://arxiv.org/abs/1906.03972.

[181] Yihan Wang, Huan Zhang, Hongge Chen, Duane Boning, and Cho-Jui Hsieh. 2020. On l_p-norm robustness of ensemble stumps and trees. arXiv:2008.08755. Retrieved from https://arxiv.org/abs/2008.08755.

[182] Gary M. Weiss. 2004. Mining with rarity: A unifying framework. *ACM SIGKDD Expl. Newslett.* 6, 1 (2004), 7–19.

[183] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *J. Big Data* 3, 1 (2016), 9.

[184] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The worst-case execution-time problem—Overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3 (2008), 36.

[185] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. 2016. Understanding data augmentation for classification: When to warp? In *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications*. IEEE, 1–6.

[186] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V., Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.0814. Retrieved from https://arxiv.org/abs/1609.0814.

[187] Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M Patton. 2015. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 4.

[188] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter. 2018. CommanderSong: A systematic approach for practical adversarial voice recognition. arXiv:1801.08535. Retrieved from https://arxix.org/abs/1801.08535.

[189] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. 2018. Accelerating the machine learning lifecycle with MLflow. *Data Eng.* 41, 4 (2018), 39–45.

[190] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic autonomous driving system testing. arXiv:1802.02295. Retrieved from https://arxiv.org/abs/1802.02295.

[191] Shichao Zhang, Chengqi Zhang, and Qiang Yang. 2003. Data preparation for data mining. *Appl. Artif. Intell.* 17, 5–6 (2003), 375–381.

[192] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. 2016. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 4480–4488.

[193] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2017. Random erasing data augmentation. arXiv:1708.04896. Retrieved from https://arxiv.org/abs/1708.04896.

Received May 2019; revised December 2020; accepted February 2021