Noname manuscript No.
(will be inserted by the editor)

Testing Machine Learning based Systems: A Systematic Mapping

Vincenzo Riccio · Gunel Jahangirova · Andrea Stocco · Nargiz Humbatova · Michael Weiss · Paolo Tonella

Received: date / Accepted: date

Abstract
Context: A Machine Learning based System (MLS) is a software system including one or more components that learn how to perform a task from a given data set. The increasing adoption of MLSs in safety critical domains such as autonomous driving, healthcare, and finance has fostered much attention towards the quality assurance of such systems. Despite the advances in software testing, MLSs bring novel and unprecedented challenges, since their behaviour is defined jointly by the code that implements them and the data used for training them.
Objective: To identify the existing solutions for functional testing of MLSs, and classify them from three different perspectives: (1) the context of the problem they address, (2) their features, and (3) their empirical evaluation. To report demographic information about the ongoing research. To identify open challenges for future research.
Method: We conducted a systematic mapping study about testing techniques for MLSs driven by 33 research questions. We followed existing guidelines when defining our research protocol so as to increase the repeatability and reliability of our results.
Results: We identified 70 relevant primary studies, mostly published in the last years. We identified 11 problems addressed in the literature. We investigated multiple aspects of the testing approaches, such as the used/proposed adequacy criteria, the algorithms for test input generation, and the test oracles.
Conclusions: The most active research areas in MLS testing address automated scenario/input generation and test oracle creation. MLS testing is a rapidly growing and developing research area, with many open challenges, such as the generation of realistic inputs and the definition of reliable evaluation metrics and benchmarks.

Keywords Systematic Mapping · Systematic Review · Software Testing · Machine Learning

1 Introduction

Humanity long dreamed about reproducing intelligence within artificial machines. Back in 1872, the novelist S. Butler was the first to describe machines developing consciousness in his work entitled “Erewhon”. Scientists

V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, P. Tonella
Università della Svizzera Italiana (USI), Via Buffi, 13 – Lugano, Switzerland
tel +41 58 666 40 00, fax +41 58 666 46 47
E-mail: {vincenzo.riccio, gunel.jahangirova, andrea.stocco, nargiz.humbatova, michael.weiss, paolo.tonella}@usi.ch

did not wait long to investigate in this direction: in 1950 Alan Turing proposed his famous operational test to verify a machine's ability to exhibit intelligent behaviour indistinguishable from that of a human [129].

The advent of Machine Learning (ML) along with recent technological advancements allowed giant steps towards the realisation of this dream. Unlike traditional software systems, in which developers explicitly program the systems' behaviour, ML entails techniques that mimic the human ability to automatically learn how to perform tasks through training examples [114]. Instances of such tasks include image processing, speech, audio recognition, and natural language processing. The huge availability of sample data, united with the increasing computing capacity of graphical processing units, has allowed training complex ML architectures which are able to outperform traditional software systems and sometimes even humans. Nowadays, it is quite common to integrate one or multiple ML components within a software system. In this paper, we refer to a system of this kind as Machine Learning based System (MLS).

MLSs are utilised in safety/business critical domains such as automotive, healthcare and finance. In these domains, it is essential to check for the reliability of an MLS, i.e., to understand to what extent they can be trusted in response to previously unseen scenarios that might be not sufficiently represented in the data from which the system has learned. To this aim, software testing techniques are being used to check the functionality of MLSs and ensure their dependability. However, the effectiveness of traditional testing approaches for MLSs is quite limited. First, part of the program logic of an MLS is determined by the data used for training, thus code coverage metrics are not effective to determine whether this logic has been adequately exercised. Second, learning processes used in MLSs are stochastic in nature, which makes it challenging to define deterministic oracles for them, since repetitions of the training phase may lead to different behaviours.

The software engineering (SE) research community is striving to tackle the problem of adequately testing the functionalities of MLSs, with the number of approaches proposed in the literature growing exponentially in recent years. Such proliferation of novel ML testing techniques demands for secondary studies, i.e., studies that review primary studies with the aim of integrating or synthesising knowledge related to this research area [107]. Systematic mapping studies (or scoping studies) are secondary studies designed to structure a research area driven by specific research questions. These studies involve a classification of the primary studies over a number of dimensions and counting contributions in relation to the categories of that classification [118]. The outcome of a mapping study is a set of papers relevant to the research questions, mapped to a classification [118, 130]. In this way, a mapping study allows to discover research gaps and trends in a research area [117].

In this paper, we present the results of a systematic mapping study we conducted about functional testing techniques for MLSs. The scope of this work is to help structure, curate, and unify the literature in this research area, and to analyse it in detail in order to help shed light on the potential of these techniques, and make them more visible and accessible [107].

We formulated 33 research questions to fulfil the goal of the study; then we proceeded by systematically collecting an initial pool of publications, and applied a number of inclusion and exclusion criteria to select the relevant primary studies. Then, we analysed the retained 70 papers in terms of the problem they address, the approach they propose and their empirical evaluation. In order to increase the validity, repeatability and reliability of our results, we designed a systematic protocol for the identification and classification of papers following the guidelines by Kitchenham et al. [106] and Petersen et al. [118]. We report all details of the research process, including also the actions taken to mitigate possible threats to validity, and we make all data collected and analyses performed during our study available [119].

The paper is structured as illustrated in Figure 1. Section 2 gives an overview about relevant ML concepts. Section 3 specifies the goal of this mapping and the research questions it aims to answer. Section 4 describes the process we followed to retrieve the relevant literature and to extract the information to answer the research questions. Section 5 synthesises and classifies the information we obtained from the literature. Section 6 summarises the weaknesses of the approaches presented in the literature and provides possible directions for future research. Section 7 provides an overview of the related work. Finally, Section 8 reports conclusions and future work.

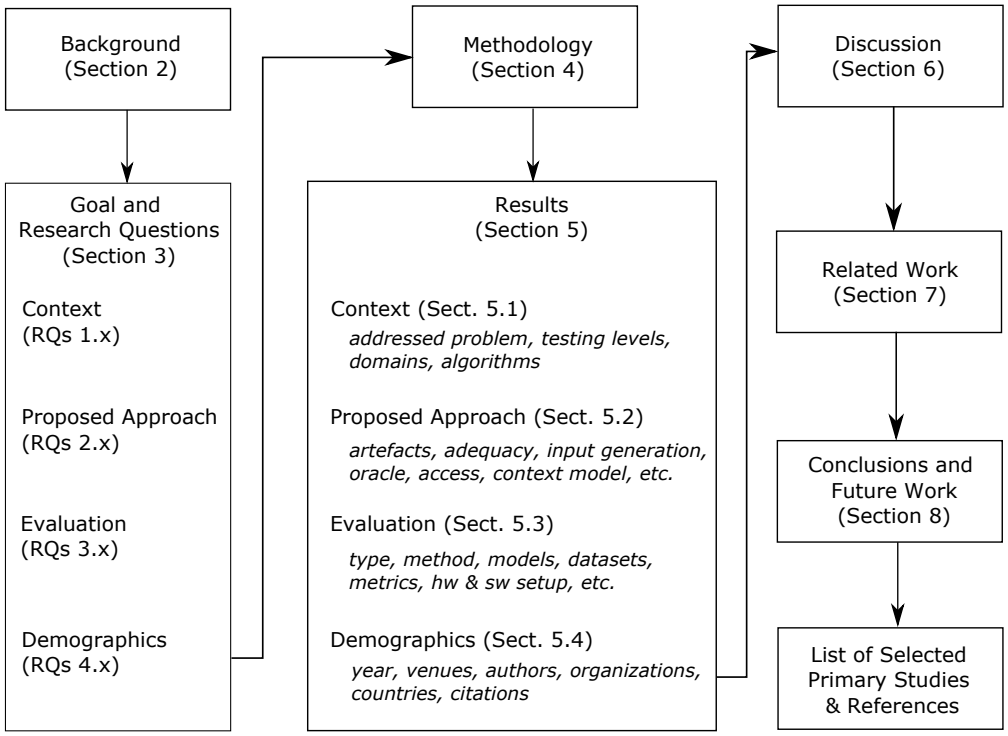

Fig. 1: Organization of the Systematic Mapping

2 Background

In this Section, we give an overview about the relevant ML techniques as well as the challenges of applying classical testing approaches in the machine learning domain.

2.1 Machine Learning Based Systems

Generally speaking, Machine Learning (ML) offers statistical techniques to learn complex functions (patterns) from a provided training data set, allowing to apply the learned functions on new, previously unseen data points. This ability to generalise is often used to make predictions about unknown properties of observed data. Once trained, ML functions can be represented and stored as a set of hyper-parameters and variables learned from the training data – a *model*, which typically includes *weights*.

Definition 1 (Model) *A machine learning model (short: model) is a trained instance of a specific machine learning algorithm.*

In real world applications, such models are typically part of a larger Machine Learning Based System (MLS). Besides one or more machine learning models, an MLS may contain also other software components, responsible e.g., for input transformation, corner-case handling, user interaction or functional logic which is not directly inferred from the training data.

Definition 2 (Machine Learning Based System) *A system in which at least one of the components relies on machine learning techniques.*

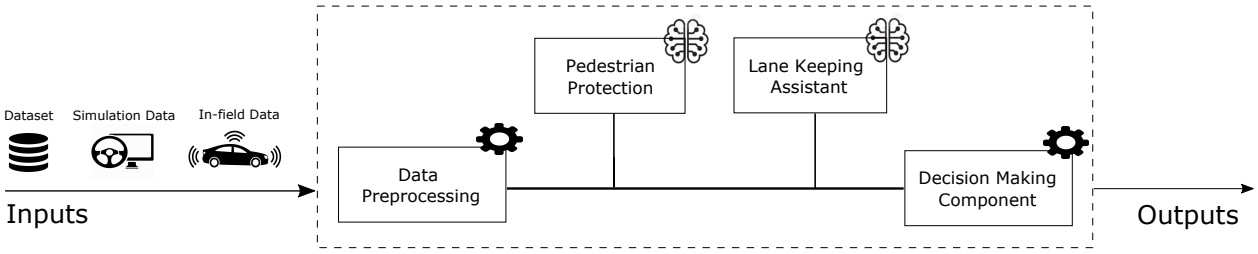

Fig. 2: Self-driving cars are representative examples of modern MLSs.

Figure 2 shows a simplified architecture of a self-driving car, which we use as an example of MLS. In the figure, the ML models (pedestrian protection and lane keeping assistant) are trained using *offline* training data (Dataset). When the whole system is in operation, the MLS takes as input *online* data, which can be either simulated or in-field input data. In our example, a data processing component is responsible of formatting and scaling the raw input data, so that they can be processed correctly by the ML models. A decision making component aggregates the outputs of various components, including the ML models, to produce the final values of the car's actuators (e.g., steering angle, brake, throttle).

2.2 Machine Learning Algorithms

Machine Learning has been the subject of research for decades. Foundations for the now very popular technique called *deep learning* were laid already in the 1940s [123]. Both research and industry are increasingly using ML techiques also thanks to the recent availability of powerful graphics processing units (GPU) at moderate prices and large amounts of labeled sample data. Researchers have investigated a wide range of algorithms, with different strengths and weaknesses. In this Section, we give a quick overview of some of the most relevant types of ML algorithms, with no claim of completeness. For a more complete introduction to ML, we refer the reader to the relevant literature [75, 102, 128].

Classification vs. Regression: Considering predictive ML problems, we often distinguish between *classification* problems in which the ML algorithm has to assign a class to a given input, from *regression* problems in which the prediction takes the form of one or more continuous values. Examples of well known algorithms for classification problems are Decision Trees [121], k-Nearest Neighbors [72], and Support Vector Machines (SVM) [83], while Linear Regression [126] and Multilayer Perceptron [94] are basic, yet very frequently used algorithms for regression problems. Note that some algorithms, such as Multilayer Perceptron or k-Nearest Neighbors, can be used for both regression and classification with slight adaptations. Classification and regression algorithms are said *supervised* learning algorithms, because they require a training data set with labeled data, where the label is the class or the continuous value to be predicted.

Clustering: Another important class of ML algorithms consists of *unsupervised* learning algorithms and among them *clustering* algorithms are the most widely adopted. Cluster analysis aims at grouping objects into clusters so that the similarity between two objects is maximal if they belong to the same cluster and minimal otherwise [105]. Popular clustering algorithms include Hierarchical Agglomerative clustering [105], K-means++ [74], K-medoids [104], and Gaussian Mixture Models [116].

Neural Networks: The one type of ML algorithm that gained most attention recently is *Neural Networks* (NN) and the associated learning techniques are named *Deep Learning* (DL). NN are inspired by the neurons in the brains of animals and consist of artificial (simulated) neurons for which an output activation is calculated based on a weighted, often non-linear aggregation of the inputs. There are various types of NN. In *Feedforward Neural Networks* neuron connections are acyclic and values are propagated in one direction, from input to output. They are useful in non-sequential predictions. Two important types of Feedforward NN are *Multilayer Perceptrons*, which consist exclusively of fully connected neural network layers, and *Convolutional Neural Networks* (CNN), which are particularly useful in image processing as their neurons can efficiently handle small subsets of related, close proximity pixels, acting as feature extractors for image portions. Differently, *Recurrent Neural Networks* (RNN) make use of stateful neurons, which can memorise the internal state reached after observing a sequence of input data. Thus, they are particularly suitable for processing of sequential data, such as natural language sentences or speech.

2.3 Measuring ML Model Effectiveness

After training an ML model, its performance is evaluated as follows:

Effectiveness Evaluation: The data available when creating the model is divided into a *training* and a *test set*, two mutually exclusive splits of the available data. Then, the training set is used to make choices about the model architecture and to train it. Lastly, the test set, which was ignored during training, is used to calculate and report the performance of the finalised model. To allow tuning of hyper-parameters and to detect problems like overfitting to the training set, a part of the training set is often defined as *validation set*. It is not used directly for training, but instead it is used to evaluate the performance of models with different hyper-parameters or to decide when to stop training (i.e., to fix the hyper-parameter named *epochs*). Training terminates when effectiveness does no longer increase if measured on the validation set. After the optimal hyper-parameters have been chosen, a final model is trained on the complete training set, including validation set.

In supervised learning, effectiveness is measured differently for classifiers and for regressors. For classifiers, *accuracy* is one of the most widely used metrics. It measures the proportion of correct classifications over all samples to be classified. For regressors, *mean squared error* is the most preferred metric, computed as the average squared difference between predicted and correct values. Unsupervised learning makes use of different effectiveness metrics, as it cannot rely on any ground truth labelling of the data. For instance, in the case of clustering algorithms, the Silhouette metric is used to get insights about the potential number and the quality of clusters within a dataset [122].

Confidence: For any non trivial problem, ML performance on the test set cannot reach the upper bound (1 for accuracy; 0 for mean squared error). In fact, perfect generalisation from training data to test data is generally impossible [89]. Moreover, both training and test data can be noisy. The probability of mis-prediction due to these reasons is named *uncertainty* and its complement is *confidence*.

2.4 Testing MLSs vs Programmed Software Systems

Testing software systems is a complex task aiming to detect and prevent unintended behavior of the software. Besides the well known challenges of classical (i.e., non-ML based) software systems, testing MLSs poses additional challenges. First, the behaviour of an MLS is largely impacted by factors such as the available training data sets, the choice of hyper-parameters, the model architecture, algorithm and optimiser. On the other hand, the source code is usually very succinct and less prone to errors, because it consists of a plain sequence of API function invocations, and the decision-making policy (i.e., the actual algorithm) is inferred from the training

data. Additionally, the exhibited behaviour, encoded by the learned weights within the model, is very difficult to interpret and debug for humans. Classical testing techniques are not easily applicable to any of the above mentioned artefacts, except for the source code. In this section, we describe the main differences between testing classical software systems vs MLS.

2.4.1 Fault and Failure

When considering unintended behavior in software systems, we distinguish between *faults* and *failures*. According to their definitions in the IEEE Standard Glossary [98]:

Definition 3 (Fault) *[. . .] An incorrect step, process, or data definition in a computer program.*

A typical example of a fault in a classical software system is a wrong instruction in a line of code.

Definition 4 (Failure) *The inability of a system or component to perform its required functions within specified performance requirements.*

To prevent failures, testing of classical software systems attempts to identify and eliminate faults. Since uncertainty is unavoidable when using ML techniques and mis-predictions are hopefully rare, but definitely possible, a robust MLS should capture such uncertainty and take suitable countermeasures to prevent failures even when an internal ML model fails to make a correct prediction. The inability of an MLS to do so can be considered as a *data sensitive fault*, according to the IEEE Standard Glossary definition [98]:

Definition 5 (Data Sensitive Fault) *A fault that causes a failure in response to some particular pattern of data.*

Exposing all data sensitive faults is a difficult task in ML applications, given their intangibly big input space. It is the main goal of MLS testing to find ML mis-predictions that give raise to MLS failures.

2.4.2 Test Input Generation

The large input space is the main root cause for faults in MLS. Thus, generating test data that represent the input space properly is an important, yet difficult task. The generated data has to fulfil various criteria, such as diversity and realism, and must be equipped with a suitable oracle.

The generation should also scale well, allowing for the creation of sufficient testing data to reliably analyse the robustness of the MLS. Depending on the domain, in practice, this may be a major challenge [80] (consider, e.g., crashing a self-driving car for testing purposes [81]). Therefore, tests are conducted within simulated environments, which enable the detection of failures in a scalable and reproducible manner [81, 127]. The generation of input data for MLS is particularly challenging since the *validity domain* (i.e., the subset of the input space that is considered a valid input) is often ambiguous and has blurred borders.

2.4.3 Adequacy Criteria

Test Adequacy Criteria aim to evaluate whether the suite of implemented tests is comprehensive enough to test the software thoroughly. Classical metrics such as code coverage are severely limited with respect to ML components, primarily as they saturate with any test suite, since the ML code is usually just a sequence of library function invocations that is fully covered when at least one test input is processed. Mutation Adequacy offers an interesting alternative and is more adequate to MLSs. It is measured as the proportion of mutations—artificial faults injected into the code—that the test suite detects (i.e., *kills*), provided ML-specific mutation operators are defined and implemented. Hence, for MLS new ML-specific adequacy criteria must be introduced, which typically either analyse the used test data or the internal states of the ML component.

2.4.4 Oracle

The effectiveness of a set of tests depends strongly on their oracle, i.e., on a way to determine whether the tested behavior is correct or a problem is observed. In the absence of oracles, tests can only detect crashes (*smoke testing*).

While conceptually oracles are the same for both classical software systems as well as MLSs (i.e., they encode the intended system behaviour), for complex domains the correct behaviour of an MLS can be challenging to define even for humans. Moreover, the nondeterministic behaviour of MLSs (i.e., repeated training on the same data may result in different behaviours) require novel notions of statistical correctness. In the context of MLS, Metamorphic Oracles have emerged as a viable approach to infer oracle information from data. Metamorphic oracles take advantage of *metamorphic relations* between input values: if a metamorphic relation holds between the inputs, the corresponding MLS outputs must necessarily satisfy a known relation (usually equality or equivalence). For instance, an image representing a handwritten digit classified as a "5" should remain in such class upon minimal addition of noise to a small number of pixels.

2.5 MLS Testing Levels

Test levels define the granularity of the tested entity, e.g., whether the tested entity is a small unit or the system as a whole. Typical test levels in classical software engineering include *unit*, *integration*, and *system* level testing. All such testing levels can also be used for MLS, but they may require further specialisation.

In this paper, we distinguish between *input testing*, *model testing*, *integration testing* and *system testing*, as defined below. Please note that while the definition of integration testing and system testing are taken from the IEEE Standard Glossary [98], input testing and model testing are new test levels we introduce to better capture the specific properties of MLS testing.

Definition 6 (Input Testing) *Input level tests analyse the training data used to train the ML components as well as the input data used at prediction time.*

Input level tests do not attempt to find faults in an MLS directly, but rather identify potential reasons for unsuccessful training in the used data, thus minimising the risk of faults due to prediction uncertainty.

Referring to our example of self-driving MLS in Figure 2, input level testing could either be performed offline or online. In the former case, it allows e.g. the detection of unbalanced training data, a common issue detrimental to the ML training process. In the latter case, online input validation can be used to identify out-of-distribution inputs, i.e., input images of driving conditions that are underrepresented in the initial training set (e.g., being taken in extreme weather conditions). The latter is a form of input validation.

Definition 7 (Model Testing) *Model level tests consider an ML model in isolation, e.g., without taking any of the other components of the MLS into account.*

Model level tests aim at finding faults due to, e.g., suboptimal model architectures, training process or model hyper-parameters.

Typical metrics used when performing model level testing include the classical measures of accuracy (for classifiers) or mean squared error (for regressors) given a labelled test dataset.

Model level testing can be considered as the equivalent of *unit testing* for units that rely on training (i.e., models). In our self-driving car example, model testing can be used to identify inputs for which the lane keeping assistant model produces wrong steering angle predictions, i.e., predictions that deviate substantially from the given ground truth.

Definition 8 (Integration Testing) *Testing in which software components (including ML models), hardware components, or both are combined and tested to evaluate the interaction between them.*

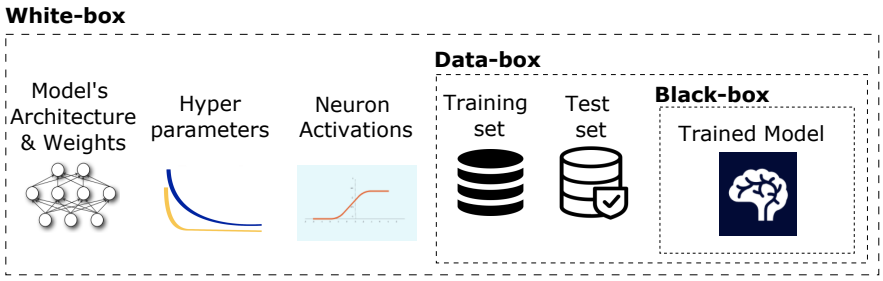

Fig. 3: Access Levels during MLS Testing

Integration testing focuses on issues that emerge when multiple models and components are integrated, although each of them behaves correctly in isolation (i.e., for which input or model level tests pass). For example, in our self-driving car, a critical integration scenario would be the case in which the decision making component must decide between possibly hitting a pedestrian suddenly crossing the road (resulting in a failure of the pedestrian protection component), or swerving and possibly crashing the car (i.e., a failure of the lane keeping assistant component).

Definition 9 (System Testing) *Testing conducted on a complete, integrated ML based system to evaluate the system's compliance with its specified requirements.*

System level testing aims to test the MLS as a whole in the environment in which it is supposed to operate. For our self-driving car example, system testing would involve running the car in realistic conditions, for instance by finding tracks and environmental conditions that are critical to handle, or by checking the behaviour in relation to other vehicles and obstacles.

System testing can be either simulated, in which synthetic input data and MLS outputs are processed in a simulation environment, or performed *in-field*, where the system is tested in the same runtime conditions experienced by the end-users.

2.5.1 Access to the Tested Component

Tests that rely only on the system's inputs and outputs are called *black-box* tests. They usually generalise well when the software architecture changes from classical software to MLS, but are limited as they cannot rely on coverage of the internal system states. Techniques that instead directly observe the program execution and the execution states are called *white-box* tests. Traditional white-box techniques relying on code coverage are quite ineffective on MLS, as coverage of the code is often trivially achieved by any test suite, regardless of its quality, because the behaviour is not encoded in the code's conditional logic (usually, the code for an ML component is just a plain sequence of instructions).

For the aim of this systematic mapping, we found it useful to introduce a new class of tests called *data-box* tests, which can be regarded as intermediate between black-box and white-box tests. The relation between black-box, data-box, and white-box testing levels is illustrated in Figure 3. More precisely, we adopt the following definitions of testing accesses:

Definition 10 (Black-Box Testing) *A black-box test has only access to the MLS inputs and outputs.*

This definition is conceptually equivalent to black-box testing in traditional software testing, as black-box tests do not rely on the internal architecture or state of the system under test. Of course, in their execution the tested systems may use an ML model to make predictions, which are then reflected in the tested output.

Inputs for black-box tests can be selected e.g. by equivalence class partitioning, boundary value analysis, or diversity.

In MLS testing, the data used to train the model (training set) and to assess its performance after training (test set) determine the behaviour exhibited by the MLS at runtime. Correspondingly, several ML testing techniques are based on the training/test data, rather than the code implementing the ML model. This introduces a new access level to the system under test that fits neither the definitions of classical black-box nor white-box testing. Therefore, in this paper, we introduce a novel testing access for MLS called data-box testing.

Definition 11 (Data-Box Testing) *A data-box test has access to everything a black-box test has access to. In addition, a data-box test makes use of the training/test data that was originally used to train/assess the ML component.*

Data-box testing is more permissive than black-box testing, even though they both do not rely on the internal architecture or state of the system under test.

Typical use-cases of data-box tests include the modification of the training set followed by a re-training of the model, in order to observe its change in prediction capability. Training sets may also be considered in isolation, to check the heterogeneity of the data and its representativeness of the complete input space. The test set can be also checked for representativeness and completeness.

Referring to Figure 2, data-box testing could for instance identify underrepresented weather conditions in the training set in order to exercise more extensively the self-driving car when running in such conditions.

Our definition of white-box testing for MLS is consistent with the classical testing literature, where the test method has access to everything within the tested component.

Definition 12 (White-Box Testing) *A white-box test has access to the internals of the tested component. This includes the ML model, its code, its hyper-parameters, its training/test data and its runtime state (e.g., neuron activations).*

Typical use-cases of ML specific white-box testing include the analysis of the diversity of the states of a model under test given some inputs (e.g., the analysis of the neuron activations in an NN when presented with the validation or test set) as well as mutation testing, achieved through ML specific mutation operators, such as changes of the model's architecture, weights, hyper-parameters, or training data.

As shown in Figure 3, black-box testing is subsumed by data-box testing, which in turn is subsumed by white-box testing, because black-box testing has access only to inputs and outputs; data-box testing has access to the training/test data in addition to inputs and outputs; white-box testing has access to the model's architecture, hyper-parameters, runtime state, in addition to the features accessible to data-box testing.

3 Goal and Research Questions

The goal of our mapping is:

> *To analyse the existing proposals for functional testing of machine learning based systems, when engineering a software system that includes one or more machine learning components.*

To reach this goal, we have analysed the approaches proposed in the literature from three different perspectives: (1) the *context* of the problem they address; (2) the *features* of the different proposed approaches; and (3) their empirical *evaluation*. Moreover, we report *demographic information* about the ongoing research. The research questions used to answer our goal are discussed in the following sections.

Table 1: Research Questions about the Context

ID	Research Question	Data Item	Values
1.1	What problems in testing MLSs are tackled?	Addressed problems	Textual description of the addressed problems
1.2	What are the testing levels addressed by the proposed solutions?	Testing levels	*Input, model, integration, system*
1.3	What domains can the considered MLSs be applied to?	Domains	Autonomous systems, image classification, etc.
1.4	Which kind of ML algorithm can the proposed solutions be applied to?	ML Algorithm	NN, clustering, classification, etc.

3.1 Context

Table 1 reports four research questions about the context of the considered studies. We want to identify the common problems in testing MLS and how they have been addressed. The answer to *RQ 1.1* will provide an overview of the problems addressed, possibly pointing to gaps in the existing literature.

With *RQ 1.2*, we want to characterise the proposed solutions by considering the testing levels to which they can be applied. We classify the MLS testing levels as input, model, integration or system as described in Section 2.5.

In *RQ 1.3*, we investigate the domains to which solutions can be applied, e.g., autonomous systems, image classification. It is important to highlight that the domains considered in the experimental evaluation of a testing solution can be a subset of all the possible ones. In this research question, we are interested in the potential applicability scope in a broader sense. Indeed, some techniques may be domain-agnostic, even though in the empirical evaluation they have been applied only to a specific domain. The evaluation domain is analysed in detail in the research questions about the evaluation (Section 3.3).

Finally, *RQ 1.4* characterises the ML algorithms to which the proposed solutions can be applied. An overview of the most important ML algorithms has been provided in Section 2.2.

3.2 Proposed Approach

Table 2 reports seven research questions that characterise the testing approaches proposed in the literature and assess their availability either as open source or closed-source. In *RQ 2.1*, we report the test artefacts generated by the approaches proposed in the literature, such as test inputs, or oracles.

RQ 2.2, *RQ 2.3* and *RQ 2.4* address three fundamental aspects of the testing solutions, i.e., the test adequacy criteria, the test input generation approach, and the test oracle. In Section 2, we have described the intrinsic challenges faced when adapting these traditional testing concepts to MLS. The answers to these three questions will provide an overview of the existing solutions to the adequacy, input generation and oracle problems tackled when testing MLS.

With *RQ 2.5*, we want to characterise the proposed approaches by the access they have to the tested component. We will classify the MLS access type as black-box, data-box and white-box as described in Section 2.5.1.

RQ 2.6 aims to investigate whether and how the testing approaches proposed in the literature leverage a model of the execution context in which an MLS operates (e.g., a model of the environment in which a self-driving vehicle drives). *RQ 2.7* concerns the availability of the proposed solutions, in order to assess the reproducibility of the research on MLS testing.

Table 2: Research Questions about the Proposed Approach

ID	Research Question	Data Item	Values
2.1	What are the generated test artefacts?	Test artefacts	Test inputs, oracle, etc.
2.2	Which test adequacy criteria have been adopted or proposed?	Test adequacy criterion	Coverage, combinatorial, diversity, etc.
2.3	What approaches are adopted to generate test input data?	Test input generation approach	Random, search-based, manual, adversarial, etc.
2.4	Which test oracles are used?	Test oracle	Misclassification, mutation killing, metamorphic, etc.
2.5	What is the access to the tested component?	Access type	*Black-box, data-box, white-box*
2.6	Is the context of the MLS modelled?	Presence of the context model	*Yes, No*. If *Yes* a brief description is provided
2.7	Are the proposed solutions available?	Availability of the solution	*Yes and open-source, Yes and closed-source, No*

Table 3: Research Questions about the Evaluation

ID	Research Question	Data Item	Values
3.1	What type of research has been conducted?	Evaluation type	*Academic, industrial, academic and industrial, no evaluation*
3.2	What kind of research method has been adopted?	Evaluation method	Experiment, experiment with humans, proof of concept, etc.
3.3	Which ML models have been considered?	ML model	Model name, e.g., ResNet50, VGG-16, Dave-2
3.4	Are the considered ML models available?	ML model availability	*Yes, No, NA*
3.5	Are the ML models already trained?	Pre-trained model	*Yes, No, NA*
3.6	What datasets have been used to train the ML models?	Dataset	Dataset Name, e.g., MNIST, Udacity challenge, ImageNet
3.7	Which systems have been considered?	System	System name
3.8	Are the systems available?	System availability	*Yes, No*
3.9	Which simulators have been used?	Simulator	Simulator name
3.10	What type of failures have been considered?	Failure	Failure type
3.11	What metrics have been adopted?	Metric	Metric name
3.12	Are there comparative studies?	Presence of a comparative study	*Yes, No*
3.13	Is the experimental data available?	Experimental data availability	*Yes, No*
3.14	What time budget has been used?	Time budget	Time budget length
3.15	What is the software setup?	Software setup	Software setup description (OS, ML framework, etc.)
3.16	What is the hardware setup?	Hardware setup	Hardware setup description (GPU, processor, memory)

3.3 Evaluation

The research questions in Table 3 analyse the empirical evaluations reported in the papers. *RQ 3.1* aims to describe the types of evaluations carried out by the considered studies. We pre-defined three types of research, i.e., academic, industrial and no evaluation. We consider an evaluation as academic if it is performed on open-

Table 4: Research Questions about the Demographics

ID	Research Question	Data Item	Values
4.1	What is the number of published studies per year?	Publication Year	Four-digit year (yyyy)
4.2	In which fora is research on MLS testing published?	Venue and venue type	Venue name (acronym) and type (journal, conference, workshop, preprint)
4.3	Who are the most active researchers in this area?	List of authors	Authors' names
4.4	Which are the author affiliations?	Organisations	Organisations' names
4.5	Which countries have produced more studies?	Countries	Country ISO3 codes
4.6	Which are the most influential studies in terms of citation count?	Citation count	Number of citing papers

source systems and in a lab setting. Differently, we deem it as industrial if the evaluation was conducted on proprietary systems within a company. It is possible that a solution is evaluated in both contexts, i.e., the evaluation type may be both academic and industrial. There is also the chance that a solution is proposed, but it is not empirically evaluated on any test subject.

RQ 3.2 enumerates the evaluation methods used to assess ML testing approaches, whereas *RQ 3.3*, *RQ 3.4*, *RQ 3.5* and *RQ 3.6* concern the ML models used by the MLS under test. In particular, *RQ 3.3* aims at finding which ML models have been used, while *RQ 3.4* investigates their public availability. *RQ 3.5* enquires whether the ML models were pre-trained, or were trained by the authors of the study. *RQ 3.6* describes the datasets that are used to train the ML models.

RQ 3.7 and *RQ 3.8* investigate which MLSs were considered as test objects in the evaluation and if they are publicly available. These questions are restricted to systems that include but do not coincide with an ML model, i.e., the model has been tested as part of a larger system, not in isolation (see Definition 7 in Section 2.5). In the latter case information about the model is already available from *RQ 3.3* and *RQ 3.4*.

Since simulation engines are extremely useful for testing safety-critical systems, *RQ 3.9* provides an overview of the simulators used in the literature. *RQ 3.10* reports the types of failures that the proposed approaches detect when exercising the objects of the experimental evaluation. *RQ 3.11* aims to describe the metrics adopted to evaluate the proposed approaches. *RQ 3.12* and *RQ 3.13* investigate the generalisability and repeatability of the evaluations, by considering the presence of comparative studies and the availability of experimental data, respectively. *RQ 3.14* gives an overview of the time budget allocated for the evaluation. *RQ 3.15* and *RQ 3.16* deal with the software and hardware configurations used in the experimental evaluation.

3.4 Demographics

The six research questions reported in Table 4 aim at providing some descriptive statistics about the ongoing research in MLS testing. With *RQ 4.1*, we look at the number of published studies per year.

In *RQ 4.2*, we identify the fora in which most of the research on MLS testing is published. We are interested in the venue type, including also studies that are not already published but whose preprint is available in open-source repositories. In the answer to this question we consider also the specific publication venues, not just the publication type.

Questions *RQ 4.3*, *RQ 4.4* and *RQ 4.5* are intended to draw a picture of the most active authors, organisations and countries in this research area. *RQ 4.6* is aimed at identifying the most influential studies. We adopt the citation count as a measure of their influence on the research community.

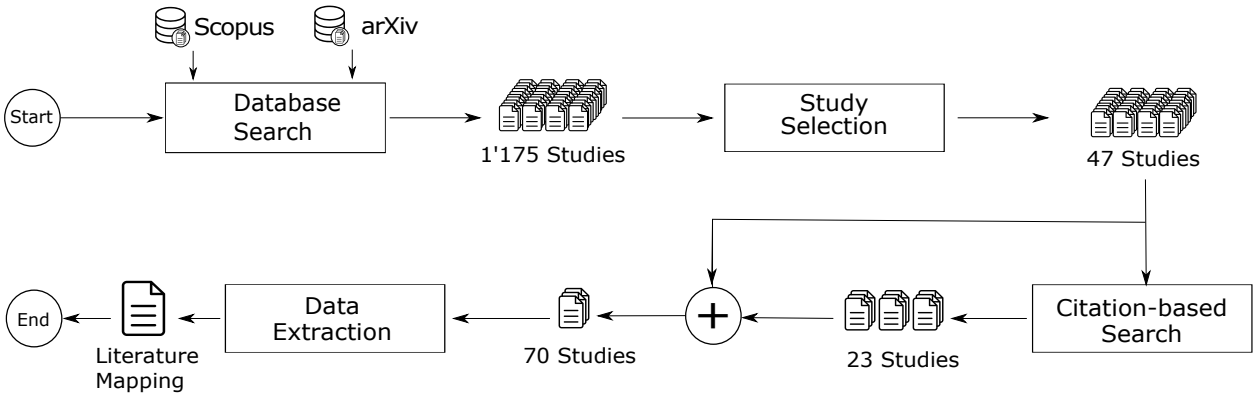

Fig. 4: Overview of the mapping process

4 Methodology

This section describes the process we carried out to obtain the relevant literature and to extract the information needed to answer the RQs introduced in Section 3. The process has been designed according to the guidelines proposed by Kitchenham et al. [106, 107] and Petersen et al. [118].

Figure 4 graphically illustrates the overall process, which consists of four main activities: (1) database search, (2) study selection, (3) citation-based search, and (4) data extraction. In the following, we refer to the authors of this paper as the *assessors*, in order to avoid confusion with the authors of the analysed primary studies.

For the database search, the assessors crafted a search string based on the goal and RQs of this mapping. They used it to query the relevant scientific databases. They also leveraged the advanced search feature usually available in these databases to automatically apply further selection criteria to the retrieved papers. The output of this step is a set of candidate relevant studies.

In the selection step, the studies obtained from the database search were assessed manually and only those studies that provide direct evidence about the RQs were retained.

Since database search and study selection could miss relevant studies, in the citation-based search step the assessors complemented the obtained pool of studies with those reached via snowballing [131] (i.e., by adding papers that cite those already included and that satisfy the selection criteria).

In the data extraction step, the assessors read and analysed in detail the relevant studies, filling out an extraction form with the information gathered from each paper.

4.1 Database Search

The database search step is aimed at finding candidate primary studies using search strings on scientific databases. In the following, we describe the databases, the search string and the selection criteria adopted by the assessors to perform the advanced search.

4.1.1 Scientific Databases

The scientific databases considered for the database search step are Scopus[1] and arXiv.[2] Scopus is a large database of abstracts and citations, containing papers published in peer-reviewed venues by multiple publishers (e.g., Elsevier, IEEE, ACM, Springer, Wiley). It offers a rich advanced search feature and is one of the suggested scientific databases to perform systematic studies in Software Engineering [107, 118].

Additionally, we considered the arXiv database to retrieve relevant papers that were not already published. This database offers an advanced search feature[3] and has been used in similar studies [76, 133] as source of grey literature. We believe that it is worth to consider grey literature since the research about testing MLS is in great ferment, with new approaches and results reported each month as preprints for early dissemination [90]. Our assumption was confirmed by the fact that out of 30 grey literature papers included in this mapping that were available only on arXiv at the time of the database search, 18 have been recently published on peer-reviewed venues.

4.1.2 Search String

The assessors adopted the following iterative process to define the search string: (1) define/refine the string, (2) perform the database search, (3) validate the results against a list of already known relevant primary studies, and (4) discuss the search results.

To formulate the string for the database search, the assessors identified an initial set of keywords starting from the goal and the research questions reported in Section 3. Each tentative search string was then validated against a list of relevant primary studies, as suggested in the guidelines [107]. This list includes papers which we were already aware of and which are expected to be included into the search results. The process terminated when the assessors were satisfied by the search results, i.e., the number of retrieved papers was manageable, all relevant primary studies known in advance were included, and no keyword relevant for our RQs was missing in the search string.

The final search string is:

> (("testing" OR "quality assurance" OR "quality assessment" OR "oracle" OR "mutation" OR "fuzzing" OR "symbolic execution")
> AND
> ("artificial intelligence" OR "machine learning" OR "deep learning" OR "neural network" OR "intelligent system" OR "intelligent agent" OR "autonomous")
> AND
> ("technique" OR "approach" OR "method" OR "methodology" OR "solution" OR "tool" OR "framework"))

The first group of terms represents the testing phase in the software lifecycle, with terms such as "testing", "quality assurance", "quality assessment", along with keywords that characterise common approaches ("mutation", "fuzzing", "symbolic execution") as well as the keyword "oracle", due to its relevance in software testing.

The second group of terms defines the MLSs that are targeted by the testing approaches. Therefore, we included keywords that define ML algorithms ("artificial intelligence", "machine learning", "deep learning", "neural network") and systems ("intelligent system", "intelligent agent", "autonomous").

[1] `https://www.scopus.com/search/`
[2] `https://arxiv.org`
[3] `https://arxiv.org/search/advanced`

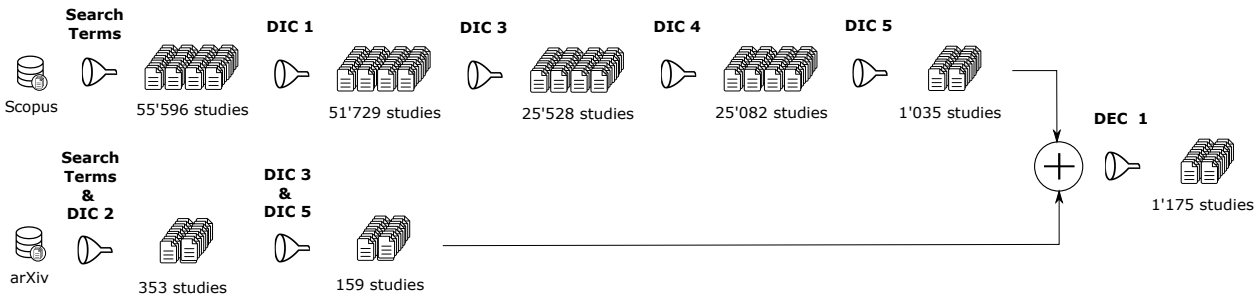

Fig. 5: Database search results

Since we are interested in papers that propose and implement ML testing approaches, the third group of terms is composed by the following keywords: "technique", "approach", "method", "methodology", "solution", "tool", and "framework".

Keywords within the same group are alternative (OR operator), but a relevant paper should present all features represented by each group. Thus, groups are connected with the boolean AND operator.

The search string was then adapted to the syntax of the considered search engines [107, 108]. In Scopus, the assessors performed this search on the metadata of the articles, i.e., the title, the abstract and the keyword list. In arXiv, they performed the search on the full text, since search on metadata is not available.

4.1.3 Database Search Advanced Selection Criteria

In the database search, the assessors applied also a set of advanced inclusion and exclusion criteria. Hereafter, we refer to them as Database search Inclusion Criteria (DIC) and Database search Exclusion Criteria (DEC), respectively.

DIC_1 *(Published in peer-reviewed journals or conference proceedings):* this criterion was applied to Scopus by adding the following condition to the search string:

```
DOCTYPE (ar OR cp)
```

DIC_2 *(Unpublished but preprint available in open access repositories):* this criterion was satisfied by performing the search on arXiv.

DIC_3 *(Subject area is computer science):* in arXiv, the assessors searched for papers in the subject area identified as `cs`. In Scopus, they applied the following condition:

```
LIMIT-TO(SUBJAREA,‘‘COMP’’)
```

DIC_4 *(English language):* it was not possible to set this criterion in the arXiv search engine. In the Scopus search, the assessors included only articles whose language is English by adding the following condition:

```
LIMIT-TO(LANGUAGE, ‘‘English’’)
```

DIC_5 *(In the Software Engineering field):* in arXiv, the assessors selected the subcategory identified as `SE`. In Scopus, they searched for papers whose venue name contains the term "software" as follows:

```
(SRCTITLE(‘‘software’’) OR (CONFNAME(‘‘software’’))
```

DEC_1 *(Published first):* published versions are chosen over unpublished ones. The assessors compared the papers based on their name and list of authors. In case of papers that were present in both databases, they retained only the one retrieved from Scopus. A further, more accurate manual filtering was carried out in the Study Selection step, since it is possible that the same paper is made available with different metadata on Scopus and arXiv.

4.1.4 Database Search Result

Both searches were performed on February 27th 2019. Figure 5 shows the results of the database search. Application of the search string reported in Section 4.1.2 retrieved 55'596 results from the Scopus database. Application of the selection criteria DIC_1, DIC_3, DIC_4 and DIC_5 narrowed down the number of primary studies to 1'035. In arXiv, we found 353 studies through the search string, which were reduced to 159 by applying DIC_3 and DIC_5.

The assessors merged the results in a shared Google sheet, obtaining 1'194 studies. They removed 18 studies by applying the exclusion criterion DEC_1 on the duplicate entries. The final number of candidate primary studies resulting from database search was 1'175.

4.2 Study Selection

Study selection was carried out by four assessors, i.e., the first three authors along with the last author. In the following, we describe the inclusion and exclusion criteria that were manually applied to the 1'175 studies resulting from the Database Search. These criteria were designed to identify primary studies that are relevant to answer our research questions [107]. Moreover, we report how the selection criteria have been applied, how many assessors evaluated each primary study, and how disagreements among assessors have been resolved.

4.2.1 Study Selection Criteria

In the study selection step, the assessors manually applied a set of selection criteria to refine the pool of papers resulting from the database search step. Hereafter, we refer to them as Manual Inclusion Criteria (MIC) and Manual Exclusion Criteria (MEC), respectively.

MIC_1 *(About MLS testing):* only studies that propose a technique to test or help testing MLS were included. The search string may have included false positives since it can retrieve studies that propose testing solutions that exploit ML techniques but their target is not an MLS.

MEC_1 *(No secondary studies):* the assessors included only primary studies, therefore they excluded mappings, literature reviews and surveys.

MEC_2 *(Remove duplicates):* the assessors kept only one copy of each study that is present in the results multiple times. This means that published versions are chosen over unpublished ones. This is similar to criteria DEC_1 applied in the database search step, but it was needed because there is a chance that the same paper is made available with different metadata on Scopus and arXiv.

Table 5: Study Selection Process

Iteration	Analysed Studies	Relevant Studies
Pilot	100	10
1	200	13
2	200	15
3	200	15
4	200	11
5	275	14
Total	1'175	78
Final Check	78	47

MEC_3 *(Extensions first):* the assessors compared the studies and chose extensions over the original versions. This criterion is applied if the two versions are either both published or both unpublished.

4.2.2 Study Selection Process

Study selection started with a meeting in which the selection criteria have been reviewed by the assessors [118]. As suggested by Ali and Petersen [71], we used a think-aloud protocol in which each assessor speaks out the thought process of inclusion/exclusion when applying the criteria to a study.

The whole process was performed in six subsequent iterations, as summarised in Table 5. The first iteration was intended as a pilot, to possibly refine the selection criteria and to ensure that they were reliably interpreted by all the assessors [107].

At each iteration, a set of studies was randomly drawn without replacement from the result of the database search. Such set was divided into two equal parts and each subset was assigned to a pair of assessors. The pairs were changed at each iteration so that, at the end of the study selection process, each study in the set was assigned to two assessors and each assessor was paired twice with all the other assessors.

The task of the assessors was to apply the study selection criteria to each study assigned to them and decide whether the study is relevant for the considered RQs. There was no single way to apply the selection criteria since it strongly depended on the analysed study. Some studies were filtered out based on titles and abstracts, while others required a full-text inspection. A careful reading was needed to determine if the systems targeted by the study were actually based on ML. In other cases, the authors had to analyse the full text to understand whether a study proposed an original testing approach rather than summarising or evaluating existing ones. There were cases in which a deeper analysis was needed to decide if the proposed approach was intended to test the MLS functionality, rather than other aspects such as security.

Each iteration was concluded with a consensus meeting in which the assessors compared their decisions. The studies for which there were conflicting opinions were discussed by the two designed assessors. In the cases in which an agreement was not found, another assessor resolved the dispute by acting as a referee.

After the last iteration, the list of relevant studies consisted of 78 papers. The four assessors checked independently these studies in a final, deeper iteration. They re-applied inclusion and exclusion criteria in a stricter way, considering the content of the candidate papers in more detail. They discussed the outcome of such further check and agreed on removing 31 papers. As a result, 47 relevant studies were selected through the described process.

The relatively low ratio between relevant and analysed studies (78 out of 1'175, or 6%) may indicate suboptimal criteria within the database search. We identified the main reason for this in selection criterion MIC_1 (About MLS testing): the database search string cannot distinguish between studies that use ML for testing and studies on testing of MLS. For such a distinction, we had to resort to manual analysis.

The final check was also quite selective, with a ratio of relevant over analysed studies around 60%. The reasons for excluding such studies were mainly: (1) a stricter application of the selection criteria, to make sure that no irrelevant paper is read and analysed in the data extraction phase; and (2) a deeper analysis of the paper content, which was not carried out in the previous phase, when a large number of papers had to be evaluated and filtered. In particular, eight studies were included in the previous stages since they propose a technique to test autonomous systems, but they were excluded in the final check because a deeper analysis of their content revealed that their target systems are not MLSs but programmed systems. Whereas, six studies that were included because they address the quality assessment of autonomous systems, were then excluded since they propose a verification technique instead of a testing one. The reasons for the exclusion of each paper during the final check are reported in the replication package accompanying this paper [119].

4.3 Citation-based Search

In this step, the first three authors and the last author complemented the results of the database search to reduce the risk of missing relevant studies. Citation-based search is a practice often referred to as *snowballing* [131], which consists of adding studies mentioned in the references of the already included papers (*backward snowballing*) or studies that cite them (*forward snowballing*). Our initial set of 47 studies identified in the study selection step was divided into four subsets, each of which assigned to one assessor. In backward snowballing, the assessors examined the papers cited by the studies assigned to them, whereas in forward snowballing, they used Google Scholar[4] to retrieve the papers citing them.

The assessors applied the selection criteria described in Section 4.1.3 and Section 4.2.1 to the snowballed papers. Relevance of the analysed papers for software engineering was taken for granted, given their presence in the citations or references of the starting set. Therefore, the assessors did not apply criterion DIC_5.

At the end of the citation-based search step, 23 papers were added and the pool of relevant primary studies grew up to 70 papers. The relatively high number of papers added by the snowballing procedure can be explained by the fragmentation of the research area. In fact, studies on ML testing are not necessarily published on software engineering venues, which delimit the scope of our database search. Other neighbouring disciplines, including machine learning itself, host papers that deal with testing issues and are relevant for our RQs. In fact, out of the 23 newly added papers, none of them belongs to the output of database search. So, while all of these papers were missed in the database search, they are indeed relevant for this mapping. This shows the importance of snowballing, especially for transversal and cross-discipline fields such as ML testing.

4.4 Data Extraction

In the data extraction step, all authors of this manuscript acted as assessors and extracted the data items needed to answer the RQs from the selected relevant studies [107].

A tabular data extraction form was used by the assessors to record the information they extracted from primary studies. In particular, each row of such form reports a study and each column corresponds to a research question. The tables in Section 3 show the data item corresponding to each research question in the third column and the values it can assume in the fourth column. It can be noticed that for some data items, values are chosen from a closed set of values, whereas others can be filled with open answers. To highlight this, we show the values that answer closed-ended questions in italics. The form includes three additional columns to collect general comments, strengths and weaknesses of each study, which we used to elaborate the discussion in Section 6.

A pilot study was performed to refine the form and gain confidence in the data extraction process. The six assessors were divided into three pairs and each pair received four papers to analyse. Data extraction

[4] `https://scholar.google.com`

was performed independently by each member of the pair on the assigned papers. The pilot was concluded with a meeting in which, for each paper, the information from the two assigned assessors was compared and disagreements were resolved either by consensus among them or arbitration by the others. Moreover, the assessors exploited the experience gained through the pilot to solve minor issues about the completeness and usability of the form. At the end of the pilot, data was extracted from the first 12 papers and the final data extraction form was available.

The rest of the papers was assigned to each of the assessors through a bidding procedure. Each assessor selected 15 papers out of the remaining 58 ones, choosing papers they felt confident about or they were willing to read. Then, papers were assigned based on the bidding and trying to balance the overall workload, measured as the sum of the lengths of the assigned papers (there was substantial length variability, especially between conference and journal papers). We also conducted a post-extraction meeting [118], in which ambiguous answers were clarified.

4.5 Threats to Validity

Descriptive Validity: Descriptive validity is the extent to which observations are described accurately and objectively [118]. To reduce this threat, a data collection form has been designed to support the recording of data. However, a poorly designed data extraction form may have a detrimental effect on the quality of the results. To mitigate this threat, the data extraction form has been carefully designed in the data extraction step. We evaluated it through a pilot that involved all the assessors and revised it by addressing the encountered issues. Another threat to descriptive validity concerns a poor recording of information in the data extraction form. We mitigated it by conducting a pilot and evaluating the information reported in the form in a post-extraction meeting, as suggested by guidelines [118].

Theoretical Validity: The choice of the scientific database to search might hinder the validity of the study, since the chosen database could miss relevant studies. We selected Scopus since it is a large database that contains papers published by multiple publishers. Moreover, to mitigate the risk of *publication bias*, i.e., the omission of relevant but unpublished studies, we included the grey literature by performing a search on the arXiv database, which is the reference repository for unpublished works in computer science. To further mitigate the risk of missing relevant studies, we complemented database search with snowballing. The reliability of the conclusions drawn could have been hindered by a bias of the researchers involved in the mapping process. For this reason, we made an extensive use of piloting and consensus meetings during the whole process. As an example, there is a potential researcher bias in the study selection step. We mitigated it by assigning a pair of assessors to each potentially relevant study and by conducting a consensus meeting to solve any conflict between them.

Repeatability: To ensure the repeatability of the results, we provide a detailed description of the followed process, including also the actions taken to reduce possible threats to validity. Repeatability is supported by the adoption of existing guidelines, such as the ones proposed by Kitchenham et al. [106] and Petersen et al. [118]. All data collected during our study is publicly available in the replication package accompanying this paper [119].

5 Results

In this section, we present a synthesis of the data extracted from the primary studies, in order to provide detailed answers to the research questions.

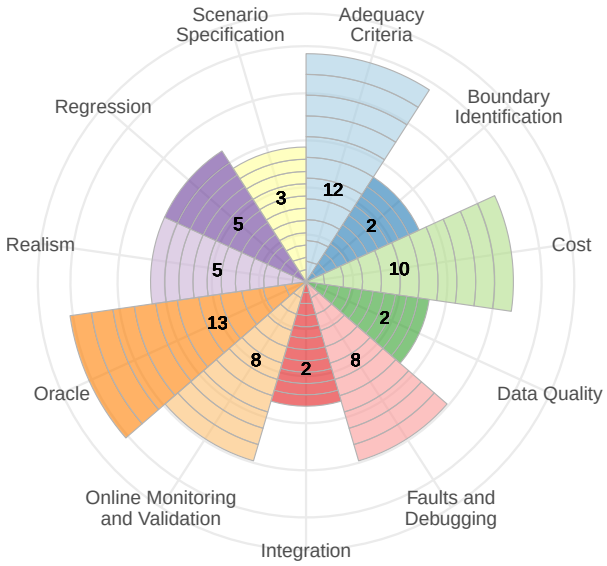

Fig. 6: Addressed Problems

5.1 Context

5.1.1 Addressed Problem (RQ 1.1)

Figure 6 presents the paper distribution across the different addressed problems. Overall, 11 main problems were identified:

Realism of test input data: Input generation should be targeted towards creating input data that can expose faults in the considered system, yet being representative of real-world scenarios [53,57]. Indeed, a fault exposed by a test input that cannot occur in practice is not a real fault. Udeshi et al. [57] propose a test input generation approach that mutates inputs in a way that makes the result conform to a given grammar, which characterises the validity domain. Tian et al. [53] produce artificial inputs that represent real driving scenes in different conditions.

A further challenge is assessing whether the results obtained in a simulated environment would also scale to the real world [24, 40, 61]. Two works propose to generate realistic test scenarios from in-field data [40], or by mining test cases from real-world traffic situations or traffic simulators [61].

Test Adequacy Criteria: Twelve papers define metrics to measure how a test suite is adequate for assessing the quality of an MLS. They often exploit them to drive test input generation. Since classical adequacy criteria based on the code's control flow graph are ineffective with NNs, as typically 100% control flow coverage of the code of an NN can be easily reached with few inputs, researchers have defined novel test adequacy criteria specifically targeted to neural networks [19, 21, 25, 26, 42, 46, 47, 51, 52, 63].

Behavioural Boundaries Identification: Similar inputs may unexpectedly trigger different behaviours of an MLS. A major challenge is identifying the boundaries between different behaviours in the input space [30, 54], which is related to boundary-value analysis in software testing [132]. For instance, Tuncali et al. [54] investigate similar scenarios that trigger different behaviours of autonomous vehicles in safety critical settings, e.g., nearly avoidable vehicle collisions.

Scenario Specification and Design: For scenario-based test cases, one fundamental challenge is related to the specification and design of the environment in which the MLS operates. In fact, only a high fidelity simulation of the environment can produce realistic and meaningful synthetic data [16, 22, 29].

Oracle: Overall, we found 13 papers in our pool that tackle the oracle problem for MLSs [10, 12, 17, 31, 33, 36–38, 43, 45, 62, 64, 70]. The challenge is to assess the correctness of MLSs' behaviour, which is possibly stochastic, due to the non-deterministic nature of training (e.g., because of the random initialisation of weights or the use of stochastic optimisers) and which depends on the choice of the training set. The vast majority of the proposed oracles leverages metamorphic relations among input data as a way to decide if the execution with new inputs is a pass or a fail, under the assumption that such new inputs share similarities with inputs having known labels [10, 12, 45, 62].

Faults and Debugging: Eight works considered in our mapping are related to faults in MLSs. Six of them address the problems of studying and defining the spectrum of bugs in MLSs, and automating the debugging of MLSs [11, 14, 15, 28, 39, 68]. Concerning the former, two studies in our pool present an empirical study on the bugs affecting MLSs [11, 68]. Indeed, the very notion of a fault for an MLS is more complex than in traditional software. The code that builds the MLS may be bug-free, but it might still deviate from the expected behaviour due to faults introduced in the training phase, such as the misconfiguration of some learning parameters or the use of an unbalanced/non-representative training set [97, 100].

Regarding debugging automation, four studies address the problem of debugging an MLS [28, 39], or localising the faults within an MLS [14, 15]. The challenge in this case is to unroll the hidden decision-making policy of the ML model, which is driven by the data it is fed with. Other two papers [23, 44] investigate how to inject faults in MLSs in order to obtain faulty versions of the system under test.

Regression Testing: Five papers deal with the regression testing problem in the context of MLSs [9, 18, 48, 60, 66], i.e., the problem of selecting a small set of test scenarios that ensure the absence of mis-behaviours on inputs that were managed correctly by the previous version of the MLS. The works by Byun et al. [9] and by Shi et al. [48] both propose a *test prioritisation* technique to reduce the effort of labelling new instances of data. Groce et al. [18] deal with *test selection* for MLSs, whereas Wolschke et al. [60] perform *test minimisation* by identifying nearly-similar (likely redundant) behavioural scenarios in the training set.

Online Monitoring and Validation: Eight works address the problem of online monitoring for validating the input at runtime. Since during development/training it is impossible to foresee all potential execution contexts/inputs that the MLS may be exposed to, it is likewise essential to keep monitoring the effectiveness of the systems after they are deployed "in the field", possibly preventing mis-behaviours when an anomalous/invalid input is being processed by the MLS.

Six of them leverage anomaly detection techniques to identify unexpected execution contexts during the operation of MLSs [5, 7, 20, 41, 59, 67], whereas two papers are related to online risk assessment and failure probability estimation for MLSs [50, 58].

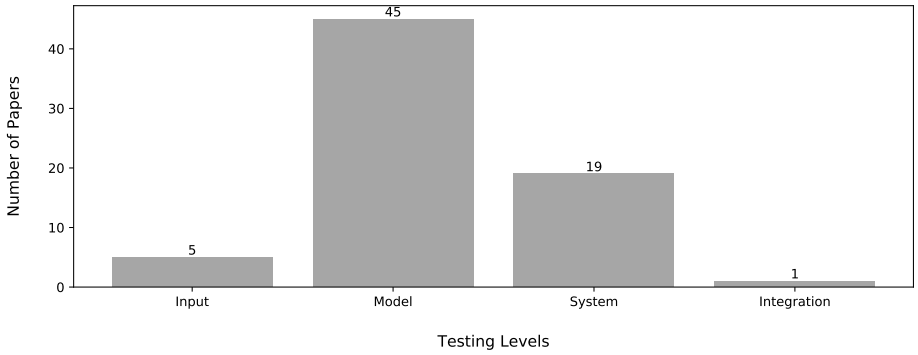

Fig. 7: Testing Levels

Cost of Testing: The cost of performing MLS testing is particularly challenging, especially in resource-constrained settings (e.g., during system or in-field testing) and in the presence of high dimensional data. Eight papers tackle this problem in the automotive domain [1, 2, 4, 6, 8, 35, 55, 69]. In this domain, comprehensive in-field testing is prohibitively expensive in terms of required time and resources. Therefore, simulation platforms are typically used to test MLSs since they allow re-testing new system releases on a large number of conditions, as well as in challenging and dangerous circumstances (e.g., adverse weather, or adversarial pedestrians suddenly crossing the road) [127].

Integration of ML models: Two papers in our pool test the interplay of different ML models within the same system [3,65]. Abdessalem et al. [3] address the functional correctness of multiple ML models interacting within autonomous vehicles. Differently, Zhang et al. [65] focus on different levels of metamorphic testing applied to two different computer vision components within a pipeline.

Data Quality Assessment: MLSs may exhibit inadequate behaviours due to poor training data, i.e., inputs that are not representative of the entire input space. At the same time, low quality test data may produce misleading information about the quality of the MLS under test. Hence, the key step towards improving the MLS quality is by achieving high training/test data quality [27, 56].

5.1.2 Testing Levels (RQ 1.2)

Figure 7 illustrates graphically the paper distribution across testing levels. Five works (7%) manipulate only the input data, i.e., they perform input level testing [7, 9, 20, 59, 61]. The majority of the papers (64%) operate at the ML model level (model level testing) [11–15, 17–19, 21, 23, 25–28, 31–39, 41–43, 45–53, 56–58, 62–64, 66–69], whereas 27% operate at the system level [1, 2, 4–6, 8, 10, 16, 22, 24, 29, 30, 40, 44, 54, 55, 60, 65, 70]. Only one work considers multiple interacting ML models at the integration level [3]. This result indicates that ML models are mostly tested "in isolation", whereas it would be also important to investigate how failures of these components affect the behaviour of the whole MLS (i.e., whether model level faults propagate to the system level).

5.1.3 Domains (RQ 1.3)

Figure 8 illustrates graphically the paper distribution across the MLS domains. Nearly half of the analysed papers (56%) propose and evaluate a technique which is domain-agnostic, i.e., in principle it may be applicable

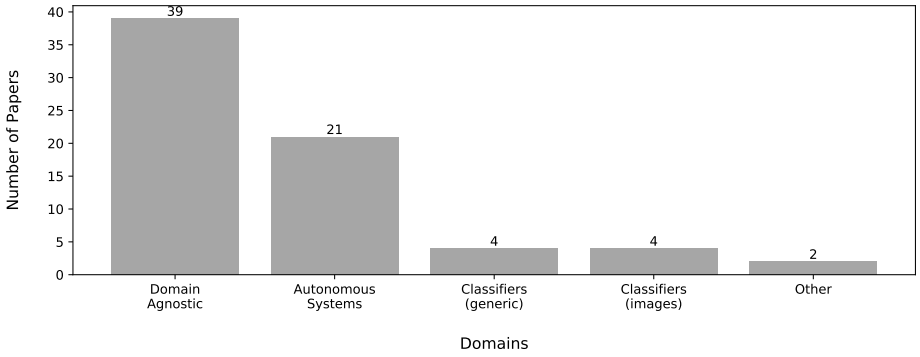

Fig. 8: Domains

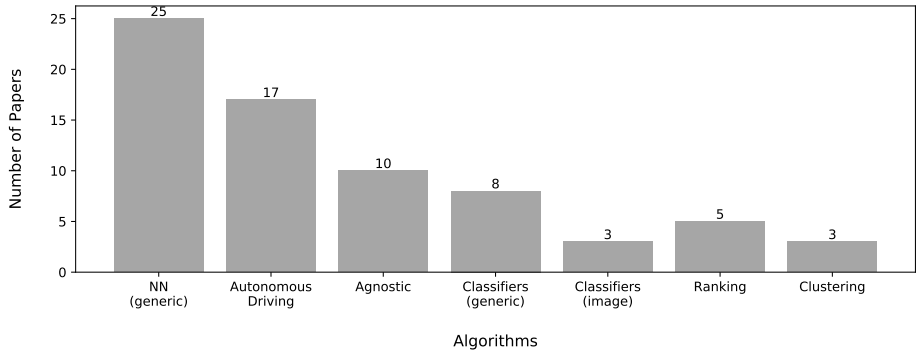

Fig. 9: Algorithms

to any MLS [5,9–11,13,15,19–21,23,25–28,31–39,42,45–48,51–53,57,58,62–64,66,68,69]. Around 30% proposed approaches are designed for autonomous systems [4, 6, 8, 16, 22, 24, 30, 40, 41, 50, 60], among which self-driving cars [7, 29, 44, 61, 67] or ADAS [1–3, 54, 55].

The prevalence of autonomous systems and in particular autonomous driving cars indicate that *safety critical domains* are those in highest demand of techniques to ensure the dependability and reliability of such systems, with testing approaches specifically designed for their peculiar features.

5.1.4 Algorithms (RQ 1.4)

Figure 9 illustrates the paper distribution across the ML algorithms to which the proposed testing solutions are applied. In some papers, the proposed technique has been applied to more than one algorithm. The majority of techniques are generically applicable to NNs (25 papers), i.e., regardless of the purpose for which the NN is used [9, 12, 13, 15, 17, 19, 21, 23, 25–28, 39, 42, 46, 47, 49, 51–53, 58, 59, 62, 65, 67]. Only one paper [13] specifically targets Recurrent Neural Networks (RNNs), which indicates that SE literature has only barely considered testing NNs related to sequential data. The second most prevalent category (17 papers) concerns autonomous driving algorithms [4, 5, 7, 16, 22, 24, 29, 30, 40, 41, 44, 50, 54, 55, 60, 61, 69]. The prevalence of NNs matches the growing popularity and success of this approach to machine learning. Since NNs are general function

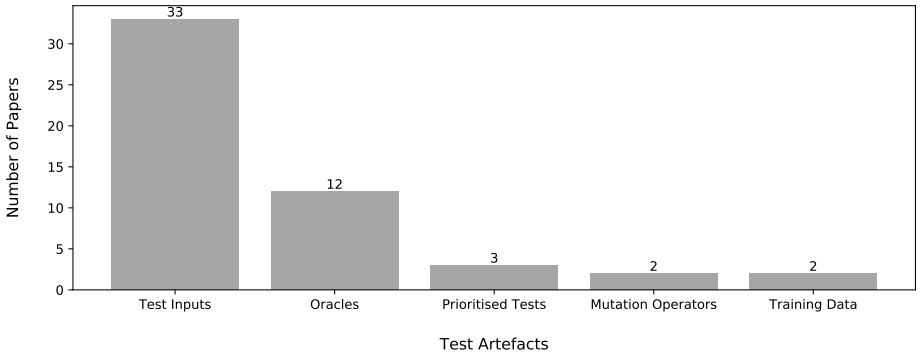

Fig. 10: Test Artefacts

approximators, they can be applied to a wide range of problems. Hence, testing techniques that prove to be effective on NNs will exhibit an incredibly wide range of application scenarios.

5.2 Proposed Approach

In the following, we present an overview of the approaches proposed in the analysed papers. We provide information on general properties of these approaches such as their generated artefacts, context model and public availability. Moreover, we focus on the specific attributes of the testing process such as the input generation method, test adequacy criteria and the oracle mechanism adopted in each of the papers.

5.2.1 Test Artefacts (RQ 2.1)

Overall, 55 out of 70 papers generate some artefact as a part of their approach. Figure 10 reports the types of artefacts that have been produced by two or more works. Nearly 60% (33 out of 55) of the papers present various methods to generate *test inputs* for the MLS under test [1–4, 6, 8, 13, 15, 16, 19, 22, 24, 25, 28, 29, 31, 32, 39, 40, 42, 46, 51–57, 60, 61, 63, 67, 70]. However, what a test input represents differs across the proposed approaches and highly depends on the domain of the tested system. As per our analysis, the most popular forms of test input are images and test scenario configurations. The inputs in form of images are generally used with classification systems or lane keeping assistance systems of self-driving cars, which aim to predict the steering angle from an image of a road taken by the camera sensor. In case the MLS under test handles scenarios with two or more interacting objects, the input for such a system is a test scenario configuration. For example, in the paper by Abdessalem et al. [3], the input of the self-driving car simulation is a vector of configurations for each of the objects involved, such as the initial position of the car, the initial position of the pedestrians, the positions of the traffic signs, and the degree of fog.

In 12 out of 55 (22%) papers the produced artefact is an oracle [12, 14, 33, 35–38, 43, 45, 62, 64, 65]. The main focus of 11 papers from this list is a set of *metamorphic relationships* (MRs), which are then used to generate a metamorphic oracle. Only one work [43] proposes a differential oracle based on program mirroring.

Compared to input generation, the oracle problem in MLS testing has received substantially less attention, indicating the need for further approaches to produce effective MLS oracles. System level oracles are particularly difficult to define, being extremely domain specific (e.g., in the self-driving car domain, they require the definition of safe driving conditions and thresholds). Moreover, they often take the form of continuous quality functions (e.g., quality of driving metrics) rather than binary ones (e.g., the car crashing or not).

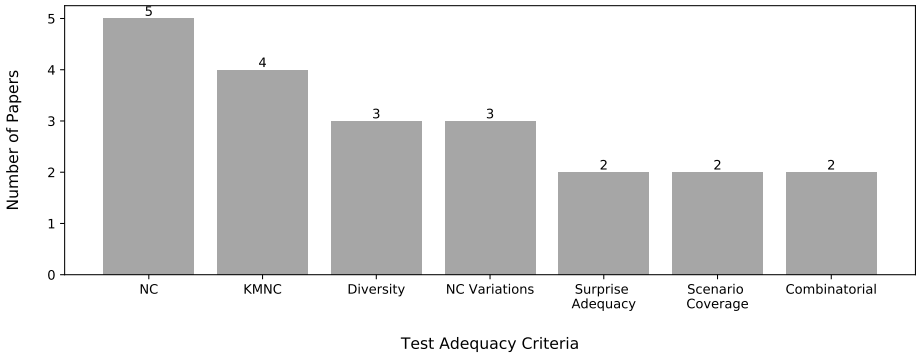

Fig. 11: Test Adequacy

5.2.2 Test Adequacy (RQ 2.2)

Test adequacy criteria have been used in 24 papers out of 70 [4, 13–15, 23–26, 30, 31, 35, 36, 38–40, 42, 43, 46, 56–58, 62, 64, 66]. Overall, 28 test adequacy criteria were used or proposed in such papers. The work by Pei et al. [42] is the first one that proposed to use neuron activations as part of an adequacy criterion. The neuron is considered activated if its output value is higher than a predefined threshold. *Neuron coverage* (NC) of a set of test inputs is defined as the proportion of activated neurons over all neurons when all available test inputs are supplied to an NN. The authors suggest that at a high level, this metric is similar to test coverage of traditional systems, as it measures the parts of NN's logic exercised by the input data.

Ma et al. [26] propose a set of five fine-grained adequacy criteria that they classify into neuron-level and layer-level. They use activation values of a neuron obtained from the training data and divide the range of values for each neuron into k buckets. The ratio of the number of buckets covered by the test inputs to the overall number of buckets (k multiplied by the number of neurons) defines the *k-multi-section neuron coverage* (KMNC). In case the activation value of a neuron is not in the range found in the training data, it is said to fall into a corner-case region. If the activation value is higher than the maximum in the range, then it is in the upper corner case region. Similarly, if it is lower than the minimum value in the range, then it belongs to the lower corner case region. *Strong neuron activation coverage* (SNAC) is defined as the ratio of the number of neurons for which upper corner cases are covered to the overall number of neurons. *Neuron boundary coverage* is defined as the ratio of the number neurons for which both upper and lower corner cases are covered to the total number of corner cases (the number of neurons multiplied by two).

Kim et al. [21] note that neuron coverage and k-multi-section neuron coverage are not practically useful, as they carry little information about individual inputs. They argue that it is not self-evident that a higher NC indicates a better input, as some inputs naturally activate more neurons. They also note that KMNC does not capture how far the neuron activations go beyond the observed range, making it hard to assess the value of each input. To overcome these limitations they propose a new metric, *Surprise Adequacy* (SA), which aims to quantify the degree of *surprise* (i.e., novelty with respect to the training set) of the neuron activation vector. Surprise adequacy has two variations: likelihood-based and distance-based. The Distance-based Surprise Adequacy (DSA) is calculated using the Euclidean distance between the activation traces of a given input and the activation traces observed during training. The Likelihood-based Surprise Adequacy (LSA) uses kernel density estimation to approximate the probability density of each activation value, and obtains the surprise of the input as its (log-inverse) probability, computed using the estimated density.

Neuron Coverage, KMNC and Surprise Adequacy are all metrics that target feed-forward DL systems. The only work that addresses coverage criteria for Recurrent Neural Networks (RNNs) is the one by Du et al. [13].

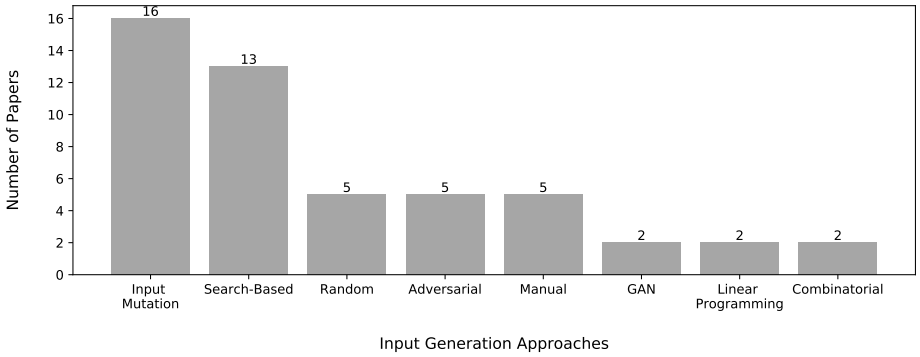

Fig. 12: Test Input Generation

In this work, authors model RNN as an abstract state transition system to characterise its internal behaviours. Based on the abstract model, they propose five coverage criteria, two of which address coverage of states and three the coverage of transitions.

Figure 11 shows how often each adequacy criterion was used. Overall, the data indicate a relatively wide adoption of the proposed adequacy criteria. Indeed, availability of ML-specific ways to measure the adequacy of the test data is crucial for MLS testing. Only a few papers adopted adequacy criteria for black box testing, e.g., scenario coverage, that is useful when we do not have white box access and we are interested in the behaviour of the whole system.

5.2.3 Test Input Generation (RQ 2.3)

Overall, 48 out of 70 papers describe how they generate inputs. As some papers use more than one input generation technique, our final list contains 52 elements, as illustrated in Figure 12.

Our analysis (see Figure 12) shows that the most widely applied technique for input generation is *input mutation* [12–14, 19, 33, 34, 37–40, 44, 45, 53, 62–64] (16 out of 52, 31%), which consists of the creation of new inputs by applying semantic information-preserving transformation to existing inputs. The majority of papers using input mutation are on metamorphic testing [12–14, 19, 33, 37, 38, 40, 45, 62–64] (11 out of 16), and the corresponding transformations are defined by a metamorphic relationship. Examples of such input mutations are affine transformations [53], change of the pixel values [37], cropping [12] for the images or alterations that mimic the environment interference for the audio files [13], designed so that they introduce changes that are imperceptible to humans. In contrast, the approach by Rubaiyat et al. [44] changes input images by simulating environmental conditions such as rain, fog, snow, and occlusion created by mud/snow on the camera. The work by Tian et al. [53] also transforms input images by mimicking different real-world phenomena like camera lens distortions, object movements, or different weather conditions. Their goal is to automatically generate test inputs that maximise neuron coverage. Similarly, the work by Guo et al. [19] has the optimisation objective of reaching higher neuron coverage, while also exposing exceptional behaviours. To achieve this goal, they mutate input images and keep the mutated versions that contribute to a certain increase of neuron coverage for subsequent fuzzing. The applied mutations have to be imperceptible for humans while the prediction of the MLS for the original and mutated input should differ (i.e., the MLS exhibits a misbehaviour).

Another widely used methodology to generate test inputs is the *search-based* approach [1–3, 6, 8, 15, 30, 42, 46, 54, 56, 57] (13 out of 52, 25%). In six papers the generation of inputs using a search-based approach aims to detect collision scenarios for autonomous driving systems. Therefore, their fitness functions use metrics such as distance to other static or dynamic objects [1–3, 8], time to collision [1, 6, 54], speed of the vehicle [3, 54] or

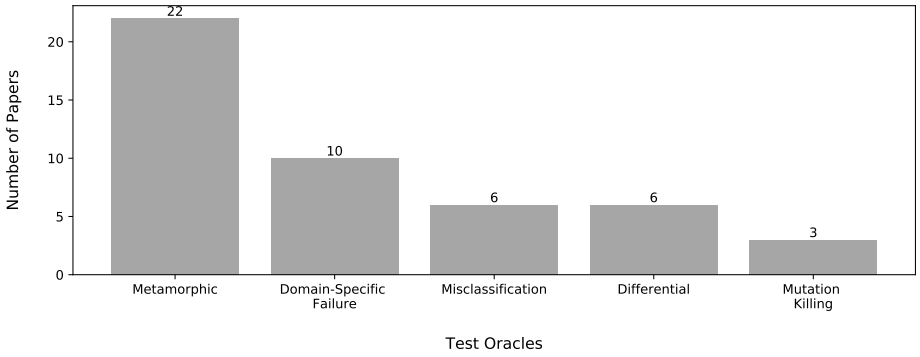

Fig. 13: Test Oracles

level of confidence in the detection of the object in front of the vehicle [1, 2]. In contrast, Mullins et al. [30] aim to identify test inputs for an autonomous system that are located in its performance boundaries, i.e., in the regions of the input space where small alterations to the input can cause transitions in the behaviour, resulting in major performance changes.

The majority of the works that use adversarial input generation (5 papers out of 52, 10%) employ the existing state-of-the-art attacking methods to generate such inputs [11, 21, 59, 66]. In contrast, the work by Abeysirigoonawardena et al. [4] has a more targeted approach which aims to create adversarial self-driving scenarios that expose poorly-engineered or poorly-trained self-driving policies, and therefore increase the risk of collision with simulated pedestrians and vehicles. While adversarial inputs can successfully trigger misbehaviours of the MLS under test, they are often very unlikely or impossible to occur in reality, unless the system is under the attack of a malicious user. However, security verification and validation of MLS is a different research area on its own and the present systematic mapping does not cover it.

5.2.4 Test Oracles (RQ 2.4)

Figure 13 provides an overview of the types of oracles that have been adopted with MLSs. The most popular type of oracle is the *metamorphic oracle*, used in 22 out of 50 (44%) papers [5,12–14,19,33–38,45,51,53–56,62–65,67]. A central element of a metamorphic oracle is a set of metamorphic relationships that are derived from the innate characteristics of the system under test. The new test inputs are generated from the existing ones using MRs so that the outputs for these inputs can be predicted. Out of 22 papers adopting a metamorphic oracle, 11 focus on proposing and evaluating novel MRs for different kinds of MLS. However, these papers mostly consider classical supervised learning algorithms, such as k-nearest neighbours, naive Bayes classifier, support vector machine, and ranking algorithms. The work by Xie et al. [64] proposes MRs for unsupervised machine learning algorithms, such as clustering algorithms. The remaining papers (11 out of 22) use MRs that are already available in the literature or that encode well-known domain-specific properties of the system.

In 10 out of 50 (20%) papers, *domain-specific failure* of the MLS under test was used as an oracle [1–4, 6, 8, 24, 39, 44, 58]. In general, failure is denoted as the weakest form of an oracle. However, only one of the analysed papers [39] conforms to such definition, i.e., the crash of the system under test. In all remaining cases, more complicated and domain-specific deviations from the expected behaviour are adopted, such as collisions with pedestrians or other vehicles, not stopping at the stop sign, or exceeding the speed limit.

Differential or cross-referencing oracle is a type of "pseudo-oracle" [84] in which multiple implementations of the same algorithm are compared against each other. If the results are not the same, then one or more of the implementations may contain a defect. This type of oracle was used in six analysed papers (12%) [31, 32,

42, 43, 46, 57]. While the work by Qin et al. [43] proposes a program synthesis approach that constructs twin "oracle-alike mirror programs", the remaining papers find different implementations for the MLS under test and use those to cross check the results. A drawback of this type of oracle is the attribution of the fault when the considered implementations produce different results. This was the case in the work by Murphy et al. [32] and the authors commented that "*there was no way to know which output was correct*". On the other hand, three papers from our pool [42, 46, 57] take advantage of such a situation, as getting different outputs in each of the different implementations makes the inputs rather interesting and worth further investigation by the developers. Pei et al. [42] and Sekhon and Fleming [46] use such differential behaviour along with a coverage criterion as part of a joint optimisation problem aimed to generate erroneous corner case scenarios for MLSs. Similarly, Udeshi and Chattopadhyay [57] propose an approach that, given a pair of ML models and a grammar encoding their inputs, searches the input space for inputs that expose differential behaviours.

Another commonly used oracle for classifiers (6 papers out of 50, 12%) is the *misclassification* of manually labeled inputs [16, 17, 25, 28, 48, 66]. While using human labels as an oracle is a pretty straightforward approach (especially for data-driven systems such as MLS), it may also require substantial effort. Another type of oracle observed during our analysis is *mutation killing*, which is used in three different papers (6%) that either propose [27, 47] or evaluate [11] mutation operators for MLS.

5.2.5 Access to the System (RQ 2.5)

The proposed testing approaches require different levels of access to the MLS under test. In 29 cases out of 70 (41%), it is enough to have a *black-box access* to the system [1, 2, 4–8, 16, 22, 24, 29, 30, 33, 35, 36, 40, 41, 43, 44, 54, 55, 57, 58, 60, 61, 64, 65, 69, 70]. In 11 cases (16%), along with the inputs and outputs of MLS, training data should also be available: black-box access is not sufficient and *data-box access* to the system should be provided [10, 12, 14, 18, 20, 32, 45, 49, 56, 62, 67]. This is mostly the case for the papers on metamorphic testing, in which the authors propose metamorphic relationships that change the training data in some specific way and then analyse the changes in the output of the retrained system. Another example of data-box access is the work by Groce et al. [18], where the distance between the training data and a test input is used as a metric to address the test selection problem. The remaining 30 cases (43%) require *white-box access* to the system [3, 9, 11, 13, 15, 17, 19, 21, 23, 25–28, 31, 34, 37–39, 42, 46–48, 50–53, 59, 63, 66, 68], as they need information on the internal state of the trained model. The most prevalent examples in this category are the approaches that need the values of neuron activations to measure some adequacy criteria [26, 42] or the weights and biases of the model to apply mutation operators on them [27, 47].

5.2.6 Context Model (RQ 2.6)

As MLSs can be complex systems that have to operate in various environments and interact with different dynamic objects, they need to be able to determine the context in which they are operating, and adapt their behaviour to context-related norms and constraints. In our pool of papers, 17 works (24%) model the context in which the MLS operates. All of them are in the autonomous driving domain [1–4, 6–8, 10, 16, 22, 24, 30, 40, 54, 55, 60, 69]. The context models presented in these papers vary in terms of simplicity and number of considered actors. For example, the work by Bühler et al. [8] addresses the problem of autonomous parking and provides a context model in which the car is the only dynamic object and the environment is represented just as a set of geometric points that define the parking space. Similarly, in the work by Beglerovic et al. [6] the goal of the ADAS is to avoid collisions and the information about the environment is provided in the form of the geometric coordinates of the static obstacles. In contrast, in the work by Abdessalem et al. [2] the authors address more complicated scenarios and their context model is much more detailed. In addition to the ADAS under test, they consider additional mobile objects such as pedestrians and other cars. Moreover, they take into account roadside objects, such as trees, parked cars, traffic signs, and describe the environment in terms of types of road, weather conditions and scene light. Considering a complex context can increase the realism of, and thus

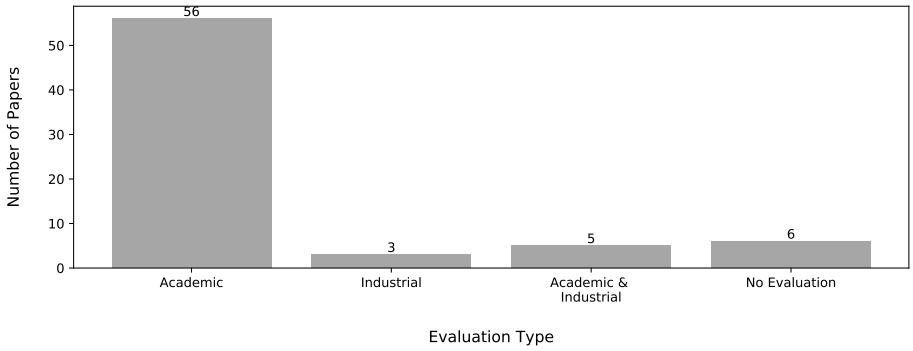

Fig. 14: Evaluation Type

the confidence on, the testing process, but it is more expensive, especially because there is no standard context model that can be reused.

5.2.7 Availability (RQ 2.7)

Availability of the proposed solutions is important in the field of software testing, as new research contributions very often rely on existing tools and prototypes. The availability of the research artefacts (i.e., experimental data, source code) of the papers on MLS testing is a strong indicator of how effectively the future research will build on and compare with the currently existing work. Our results do not draw an optimistic picture on this issue, as for 50 out of 70 (71%) papers there is no available artefact. For 20 papers (29%), the proposed solutions are available in open-source format [1–4, 6–8, 10, 16, 22, 24, 30, 40, 41, 50, 54, 55, 58, 60, 69], with the exception of one [70] where the tool is available, but the source code is not. It is worth to note that all such papers were published between the years of 2017 and 2019, which may indicate a growing positive trend toward open research in MLS testing.

5.3 Evaluation

In the following, we describe the empirical evaluations of the testing solutions proposed by the analysed primary studies. We provide an overview of the evaluation types and methods, as well as information about the objects and the experimental setup.

5.3.1 Evaluation Type (RQ 3.1)

We split the analysed studies based on the type of experimental evaluation: academic or industrial. As shown in Figure 14, the vast majority of the studies (56) carry out an evaluation in an academic context. Three studies [8, 9, 70] perform an evaluation on proprietary systems belonging to an industrial partner. Five works [1–3, 31, 34] adopted a combined approach, featuring an evaluation in both academic and industrial contexts, while six works [5, 36, 60, 61, 65, 69] contained no empirical evaluation of any sort. Taking into account that ML is widely used in industry and its applications are growing exponentially, the collected data suggest that industrial research should be given greater focus.

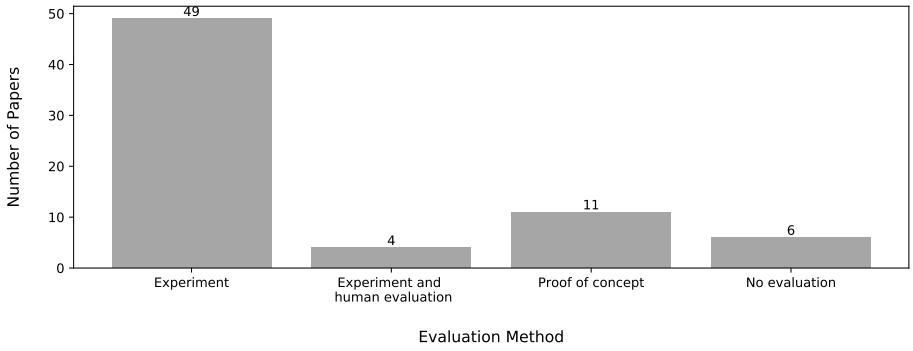

Fig. 15: Evaluation Methods

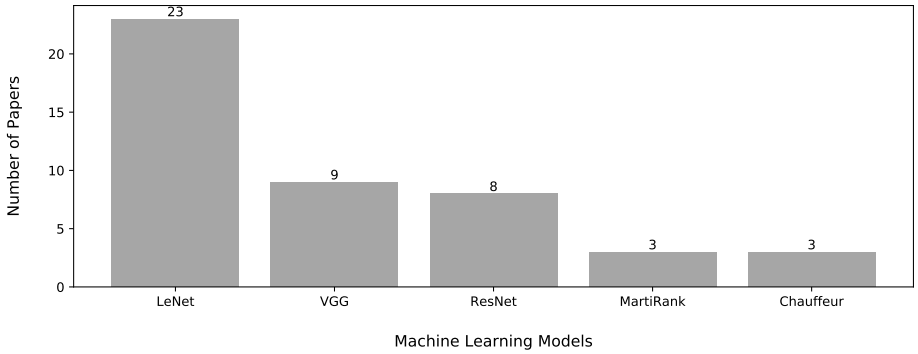

Fig. 16: Machine Learning Models

5.3.2 Evaluation Method (RQ 3.2)

We observed a number of different evaluation methods, including experiments with no human involved, experiments with human evaluation and proof of concept studies. The category representing the experimental approach with no humans is the widest one and features 49 works, which is 70% of the total number of papers. Eleven papers (16%) [7, 10, 16, 22, 24, 29, 34, 38, 46, 54, 55] provided small-scale exemplary proofs of viability for the proposed approach and are united under the "proof of concept" category. An instance of such an evaluation method is the work by Klueck et al. [22], where the authors provide an example of their test case generation process focusing only on one selected ontology and its conversion into a combinatorial testing input model. Four studies out of 70 (6%) [1–3, 18] included human evaluation in their empirical approach, while the remaining six (9%) [5, 36, 60, 61, 65, 69] carried out no evaluation of any kind (see Figure 15). Overall, the young discipline of ML testing seems to have already adopted a quite demanding evaluation standard, targeting the empirical method of controlled experiments. When operating at the system level, experiments with humans become critical to assess the validity and severity of the reported failures.

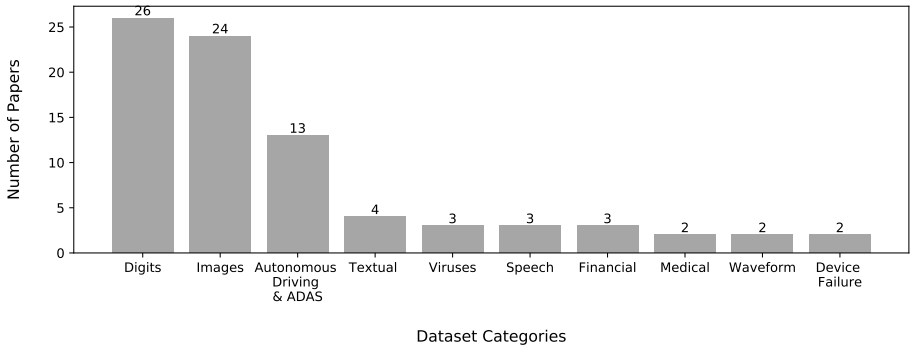

Fig. 17: Datasets by categories

5.3.3 ML Models (RQ 3.3, RQ 3.4, RQ 3.5)

We have analysed the usage of existing ML models for the purpose of evaluating the proposed testing approaches. We found that 43 out of 70 papers (61%) contain a mention of the adopted ML models. Most of them are publicly available; only 26% of these papers used models that are not open-source.

Figure 16 depicts the most popular models (to keep the size of the picture small, we report only the models that were used at least three times in the evaluation of the retrieved papers; the full list is available in our replication package [119]). Our results show that authors tend to reuse widely adopted and open-source models, such as LeNet [110] VGG [125], or ResNet [95]. In three studies [32, 33, 35], Murphy et al. used Martirank [91], a ranking implementation of the Martingale Boosting algorithm [112].

Among the steering angle prediction models for self-driving cars, five papers [21, 29, 42, 53, 67] trained their models using the datasets provided by Udacity.[5] Udacity is a for-profit educational organisation that helps students expand their machine learning skills and apply them to the area of autonomous driving. Udacity contributed to an open-source self-driving car project by releasing a self-driving car simulation platform, and by introducing a series of challenges related to various aspects of autonomous driving (e.g., lane keeping, object localisation and path planning). According to our results, the model by the team Chauffeur[6] is the most used (3 papers) [21, 53, 67].

In 58% of the cases, researchers did not use pre-trained models, but rather performed the training themselves. It can be noticed that the most widely used architectures are convolutional networks, which in turn suggests that image recognition is a frequent subject for studies and scientific experiments.

5.3.4 Training Dataset (RQ 3.6)

We studied the datasets that were used to train the ML models considered in the evaluation of the proposed approaches. Out of the 70 relevant studies, 28 (40%) did not use or did not mention the usage of any specific dataset. The remaining 42 (60%) studies mention 48 datasets of different size and nature among which 10 are custom ones. According to the content, these datasets can be classified into a number of broad categories, illustrated in Figure 17. We report only categories that contain two or more instances. We can notice that the digit recognition task is the most frequently tackled one, closely followed by the general image classification

[5] `https://github.com/udacity/self-driving-car/tree/master/datasets`

[6] `https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur`

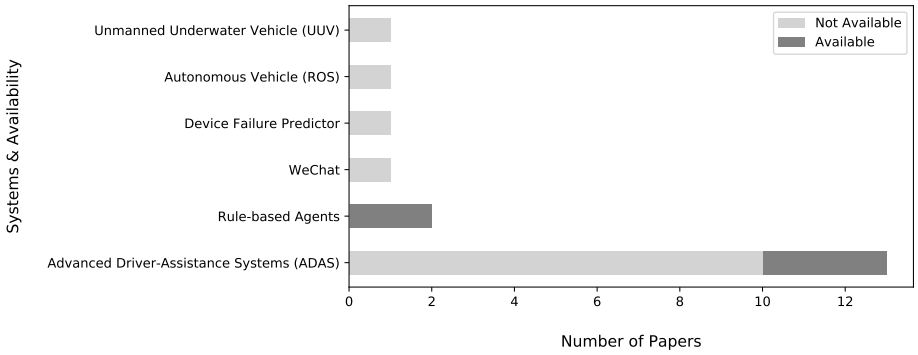

Fig. 18: Systems & Availability

problem. Among the remaining categories, the numbers of datasets used for tasks related to autonomous driving indicate a growing interest for the training of NN-based autonomous vehicles.

Concerning the datasets used, MNIST [111] (a labeled dataset of hand-written digits that has a training set of 60,000 examples, and a test set of 10,000 examples) is the most popular (29%), which is not surprising due to the frequent occurrence of digit recognition in our evaluations. CIFAR-10 [109] (a dataset of colour images in 10 classes that has a training set of 50,000 images, and a test set of 10,000 images) is also quite adopted (16%). In the autonomous driving and ADAS domains, the datasets of real world driving images released by Udacity are the most used (7%). Among the seldom used datasets are the ones targeting more specific domains of application, such as Drebin (Android malware) [73], Cityscapes (set of stereo video sequences recorded in street scenes) [82] or the UCI (University of California, Irvine) ML Repository: Survive (medical) [87]. Moreover, we noticed that the creation of a custom and tailored dataset is also a relatively frequent practice (21% of all datasets used).

5.3.5 System & System Availability (RQ 3.7, RQ 3.8)

A relatively small subset (27%) of the papers study the behaviour of the whole MLS [1–4, 6, 8, 10, 16, 24, 29, 30, 34, 40, 44, 50, 54, 55, 58, 70]. From the data in Figure 18, we can notice that advanced driver-assistance systems (ADAS) are the most widely evaluated MLSs, along with other types of autonomous vehicles (ROS) and unmanned underwater vehicles (UUV). This confirms the high interest of the academic and industrial communities towards autonomous systems.

For what concerns availability, 74% of the systems used in the relevant literature are closed-source. The largest proportion of systems that are open source consists of advanced driver-assistance systems that are contained in the ADAS category.

5.3.6 Simulator (RQ 3.9)

Nearly one fourth of the analysed studies (27%) [1–4, 6, 8–10, 16, 24, 29, 30, 40, 44, 50, 54, 55, 58, 67] make use of a simulator. In such cases, experiments are conducted in a computer simulation of the physical environment, to avoid the costs and issues associated with experiments conducted in the field, as well as to reduce the time necessary to run the experiments. Only one study [24] independently developed a custom simulator that specifically suits the requirements of the study, while the others adopted and, in some cases, modified the existing simulators.

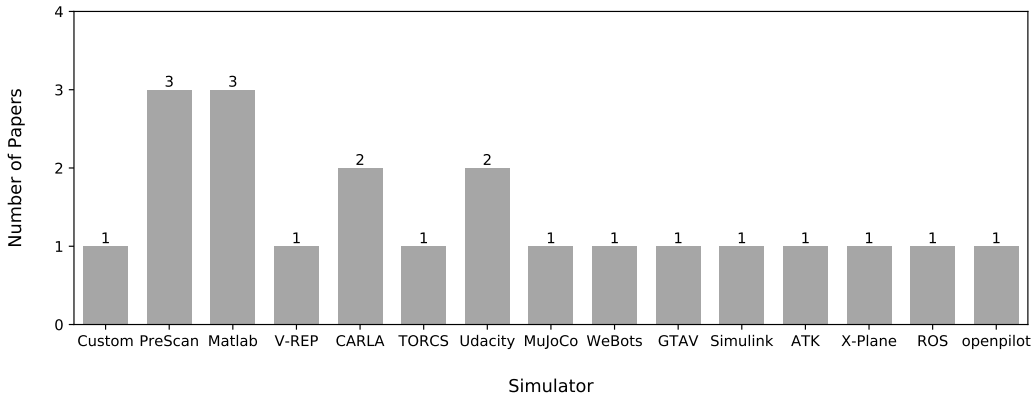

Fig. 19: Simulator

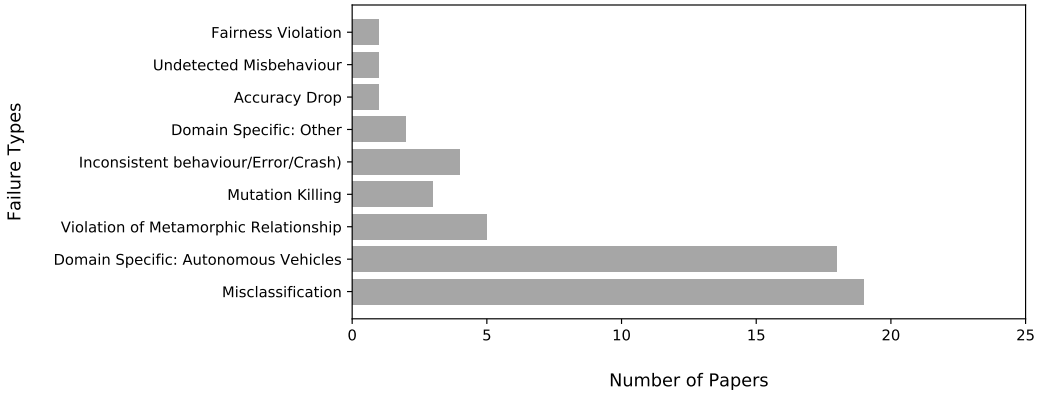

Fig. 20: Failure Types

From the adopted solutions (see Figure 19), the PreScan [99] and Matlab simulation platforms stand out in terms of the number of mentions, each being used by three works [1–3], [6, 8, 54], followed by the Carla [86] and Udacity[7] simulators, used by two works [4, 10], [29, 67]. The latter simulators are designed specifically for autonomous driving systems. Despite the prevalence of simulators tailored for ADAS and autonomous driving systems, in the list we can see a flight simulator for unmanned aerial vehicles (X-Plane 11) [9] and a robot simulator (V-REP, now discontinued and replaced by its successor CoppeliaSim) [50].

5.3.7 Failure Type (RQ 3.10)

An important aspect in the evaluation of any testing solution is the definition of *failure*. While for 18 (26%) [5,7, 10,14,17,20,22,36–38,40,49,59–61,65,68,69] papers this information is not applicable or not available, 52 (74%) works provide a description of the failure types detected in the experimental evaluation. In case the object of evaluation is an ML model, the failure type is defined based on the type of prediction (e.g., misclassification for

[7] `https://github.com/udacity/self-driving-car-sim`

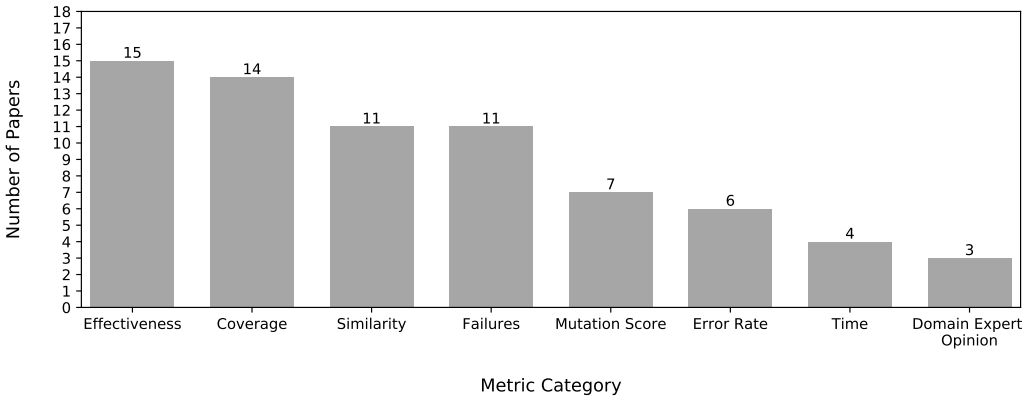

Fig. 21: Evaluation Metrics

a classifier). As the focus is shifted from isolated ML components to MLSs such as autonomous vehicles, the failure type frequently involves the choice of domain-specific parameters (e.g., the amount of deviation from the center line for a lane keeping assistance system).

In line with the results about *Training Dataset* (Section 5.3.4) and *ML Models* (Section 5.3.3), one of the most frequent failure types is misclassification. The second most popular category is domain-specific failures defined for autonomous vehicles and driving assistance systems. This set of failures combines several specific instances, the most frequent being deviation from the expected steering angle (five papers [28,41,42,53,67]) and number of collisions (10 papers [3, 4, 6, 8, 24, 29, 44, 54, 55, 58]). The importance of metamorphic and mutation testing for MLS is reflected in the relatively large number of mentions of the associated types of failures: five papers [33, 35, 45, 62, 64] used the violation of metamorphic relationships, whereas three papers [11, 27, 47] used the notion of mutation killing. The full picture is shown in Figure 20.

5.3.8 Metrics (RQ 3.11)

In this section, we discuss the metrics most frequently adopted to evaluate the proposed approaches. The exhaustive list, obtained from 60 papers (85%), covers a wide range, depending on the task considered in the evaluation as well as the testing approach and can be divided into eight main categories: (1) Effectiveness, (2) Coverage, (3) Similarity, (4) Failures, (5) Mutation Score, (6) Error Rate, (7) Time, (8) Domain Expert Opinion. The categories are listed in descending order with regards to the number of papers that used such a metric in the evaluation process and leave aside a number of metrics that were used only once or twice or are too specific to be classified. The numbers for each category are presented in Figure 21.

In the Effectiveness category, metrics based on the loss and accuracy are the most often adopted ones (ten papers [9, 12, 15, 21, 23, 28, 50, 57, 59, 63]), while precision, recall and F-measure appear in three works [16, 30, 70] and AUROC in only two papers [21, 59]. Coverage is the second most extensive class (14 papers [10, 13, 19, 21, 25, 26, 30, 40, 42, 46, 51–53, 63]), in which we can distinguish a relatively large family of neuron coverage metrics [19, 21, 26, 42, 46, 53, 63]. The variety of metrics stemming from neuron coverage are reviewed in details in the Section 5.2.2. Category Time includes execution time or time spent to generate a desired number of inputs. It served as a metric for performance evaluation in four papers [19, 48, 56, 66]. Three studies [1–3] use domain expert opinions as qualitative indicators to evaluate the proposed approaches. The time performance of an MLS testing approach is particularly important when dealing with complex MLSs, such as self-driving cars, because even in a simulation environment the budget of available system executions is severely limited.

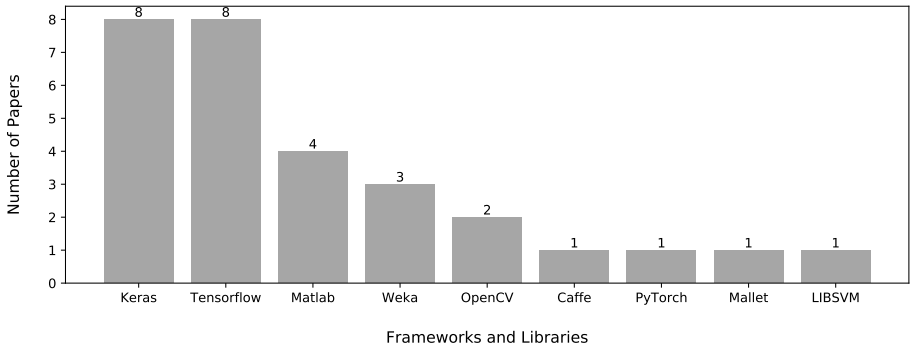

Fig. 22: Software setup

Human feedback is also quite important, since failure scenarios might be useless if falling outside the validity domain.

5.3.9 Comparative Study (RQ 3.12)

In general, comparative studies are important as they show the differences (e.g., pros and cons) between a novel technique and the state of the art. They can guide researchers and practitioners in the choice of an appropriate solution for their specific needs. In our list of relevant papers, 26 (37%) [4, 8, 9, 11, 12, 16, 18, 19, 21, 25, 26, 28, 30, 45, 46, 48, 49, 51, 52, 54–57, 62, 66, 70] include a comparative evaluation. Test input selection, generation and prioritisation are the techniques most frequently involved in comparative evaluations. For other MLS testing techniques, comparative studies are generally lacking, which may be related to the scarce availability of baseline MLS testing solutions as open-source tools (see results for RQ 2.7).

5.3.10 Experimental Data Availability (RQ 3.13)

Availability of experimental data is crucial for replication studies and for projects that build on top of existing solutions. Unfortunately, only a fraction of authors of the considered studies made their experimental data publicly available (despite we considered the cases when data is partially accessible as "available"). This fraction comprises 13 papers out of 70 (19%) [3, 9, 15, 20, 21, 42, 48, 51–53, 56, 57, 68]. This is a rather negative result, which may hinder the growth of the research in the field.

5.3.11 Time Budget (RQ 3.14)

Training and testing of MLSs is known to be a very expensive and time consuming process and getting a rough estimate of the average time spent to conduct an experiment in the field is indeed quite useful. However, only 8 (11%) [2,3,13,28,30,52,54,63] studies report information about time budget, ranging from 1/2h to 50h, with an average of 16.4h. This values give an order of magnitude of the time budget allocation necessary when testing complex MLSs.

5.3.12 Software Setup (RQ 3.15)

Concerning the software libraries and frameworks used for the implementation and evaluation of the proposed approaches, only 22 (31%) papers [6, 8, 9, 12, 15, 18, 21, 26, 27, 35, 39, 42, 45, 48, 49, 54–56, 62–64, 67] provide such

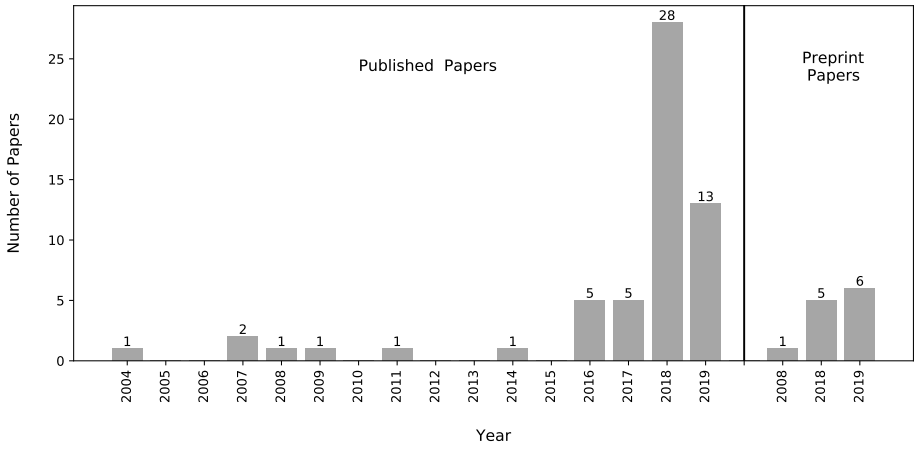

Fig. 23: Distribution of papers per year

information. As illustrated in Figure 22, Keras and Tensorflow are the most used frameworks, followed by Matlab. The Ubuntu operating system was explicitly mentioned in six papers [9, 15, 19, 35, 42, 56].

5.3.13 Hardware Setup (RQ 3.16)

Only 19 (27%) papers [7, 9, 12, 15, 16, 19, 21, 26, 27, 35, 42, 43, 48, 49, 51, 52, 56, 63, 64] contain information on the hardware setup used to conduct the experiments. In three works [26,27,49], experiments were run on a cluster of computers, while nine mention the specific GPU model that was used [7,9,12,16,19,26,27,42,63]. Interestingly, all of the GPU models reported in the papers are NVIDIA products, with the GeForce series being mentioned five times [7, 9, 16, 19, 42] and the Tesla series four times [12, 26, 27, 63]. We conjecture this result is influenced by the adoption of CUDA as parallel computing platform and programming model in the NVIDIA graphical processing units (GPUs), because Tensorflow supports NVIDIA GPU cards with CUDA.

5.4 Demographics

In the following, we report and comment some statistics about the papers considered in this study, including year of publication, venue, authors, affiliations and affiliation countries. Note that for preprint versions that were later published at a conference, journal or workshop, we always refer to the latter version. The reported data, as well as the number of citations, were collected from Google Scholar on May 11th, 2020.

5.4.1 Year of Publication

When analysing the year of publication of the papers considered in this study, we distinguish between papers that have been made available exclusively in preprint archives and papers that have been accepted for publication in a peer-reviewed journal, conference or workshop. The aggregated publication years are shown in Figure 23. The trend apparent from this figure is that the number of papers is rapidly increasing in recent years, showing a growing interest and attention of the software engineering community towards testing MLSs.

Table 6: Venues represented in the mapping (number of papers shown within brackets)

Conferences (40)			**Workshops (10)**	**Journals (8)**	**Preprint (12)**
ASE (6)	APSEC (1)	PRDC (1)	ISSREW (3)	TSE (2)	arXiv (11)
ICSE (4)	ATVA (1)	SANER (1)	QRS-C (2)	JSS (2)	CUCS report (1)
ISSTA (4)	FASE (1)	SEFAIS (1)	FAACS SEFM-W (1)	CVT (1)	
ESEC/FSE (3)	IROS (1)	SOSP (1)	ICSE-Nier (1)	IJES (1)	
AITest (2)	ISSRE (1)		MET (1)	IV Transactions (1)	
ICRA (2)	ISSSR (1)		RT (1)	SAE Tech. Papers (1)	
IV Symposium (2)	ITSC (1)		SOFL+MSVL (1)		
QRS (2)	PLDI (1)				
SEKE (2)	PMLR (1)				

5.4.2 Venue and Venue Type

Table 6 shows the publication venues of the papers considered in this systematic mapping. More than half of the papers (40 out of 70) were published at conferences; only a relatively small number at workshops (10) or journals (8). The high number (30) of papers that we found only in arXiv at the time when we downloaded all relevant works (February 27th, 2019), some of which (18/30) were published later in a peer-reviewed venue, as well as the high number of papers published at conferences/workshops, indicate the importance of fast knowledge transfer in a quickly evolving field like ML testing. It is common practice for researchers working on ML testing to continuously check for new arXiv submissions relevant for their research.

5.4.3 Authors

We aggregate statistics about the authors of the considered papers without taking the order of authors into account. Overall, 241 distinct authors contributed to 70 analysed papers; the average number of authors per paper was 4.34. On average, an author contributed to 1.26 of the papers, where 205 authors contributed to one paper, 18 to two papers and 11 to three papers. The following six authors contributed to more than three (3) papers:

6 papers: Kaiser, G. (Columbia University) [31–35, 62]
6 papers: Murphy, C. (Columbia University) [31–35, 62]
5 papers: Liu, Y. (Nanyang Technological University) [25–27, 63, 64]
4 papers: Li, B. (University of Illinois at Urbana–Champaign) [25–27, 63]
4 papers: Ma, L. (Harbin Institute of Technology) [25–27, 63]
4 papers: Xue, M. (Nanyang Technological University) [25–27, 63]

5.4.4 Affiliations

Overall, the authors who contributed to the papers considered in this study work for 84 distinct organisations.[8] On average, each of these organisations contributed to 1.68 papers and each paper was written by authors from 2.0 different organisations. The six organisations which contributed to the most papers are:

8 papers: Columbia University [31–35, 42, 53, 62]
6 papers: Nanjing University [11, 43, 47, 48, 62, 64]
6 papers: Nanyang Technological University [13, 25–27, 63, 64]
5 papers: Carnegie Mellon University[9] [17, 25–27, 63]
5 papers: Harbin Institute of Technology [13, 25–27, 63]
5 papers: Kyushu University [13, 25–27, 63]
5 papers: University of Illinois at Urbana-Champaign [25–27, 63, 70]

[8] For the analysis of authors' affiliations, we only considered the affiliations mentioned in the papers.
[9] Including Carnegie Mellon University at Silicon Valley.

Table 7: Countries (ISO3) of authors affiliations
(number of papers shown within brackets)

Asia (29)	**Europe (19)**	**America (33)**
CHN (20)	DEU (6)	USA (30)
JPN (9)	GBR (5)	CAN (3)
SGP (8)	LUX (3)	BRA (1)
IND (2)	SWE (2)	
TUR (1)	AUT (1)	**Oceania (6)**
KOR (1)	NOR (1)	AUS (6)
VNM (1)	CHE (1)	

For-Profit Organisations: It is notable that besides universities, we also observed various contributions from for-profit companies. This is particularly evident for papers in the automotive domain. For-profit organisations that contributed to papers in this domain include: IEE S.A. Contern (Luxembourg) [1–3], AVL List (Austria) [6, 22] Volkswagen (Germany) [7] DaimlerChrysler (Germany) [8] and Toyota (USA) [55]. This finding is encouraging, but we argue that more industrial involvement should be actively promoted, especially by non-profit organisations (e.g., through collaborations with the industrial sector), because of the growing number of industrial products/services that embed some ML technology and demand for dedicated ML testing techniques. Insights from the industry can help researchers steer their work towards applicable and relevant topics that can be applied in practice. Moreover, industrial data sets are crucial to evaluate the proposed techniques in realistic and relevant contexts.

5.4.5 Countries

We analysed the number of papers by country, considering the country of the affiliation indicated in each paper. If a paper was written by multiple authors from different countries, we counted that paper for all represented countries. As above, we did not take the author order into account. Table 7 shows the number of papers by country. At least one author of 31 papers was affiliated to a research institution in the United States of America (USA), more than any other country. USA is followed by China (20 papers), Japan (9 papers), Singapore (8 papers) and Australia (7 papers). Germany (6 papers) is the most active European country. Table 7 reports the paper distribution by continent. The most active continent is America (33 papers), followed by Asia (29 papers) and Europe (19 papers).[10]

5.4.6 Citation Counts

We collected the citation counts from Google Scholar on May 11th, 2020. If multiple versions of a paper were available, we aggregated the citation counts. On average, the considered papers were cited 37.62 times with a median of 13.5 citations. The most cited papers in our pool are the following:

377 citations: *DeepXplore: Automated Whitebox Testing of Deep Learning Systems* by Pei et. al. [42]
354 citations: *DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars* by Tian et. al. [53]
159 citations: *Testing and validating machine learning classifiers by metamorphic testing* by Xie et. al. [62]
113 citations: *Properties of machine learning applications for use in metamorphic testing* by Murphy et. al. [33]
111 citations: *DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems* by Zhang et. al. [67]

[10] As papers with authors from multiple countries of the same continent are only considered once in the per-continent aggregation, the number of papers per continent does not necessarily equal the sum of the number of papers by countries of that continent.

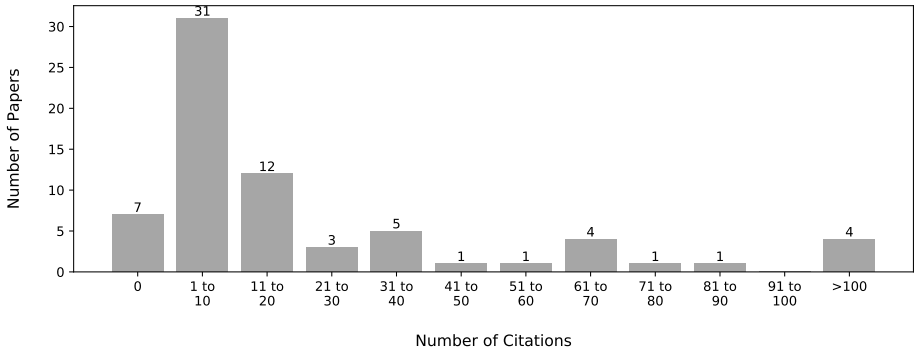

Fig. 24: Distribution of citations per paper

Figure 24 illustrates the distribution of the number of citations per paper. It is quite remarkable that the two most cited papers were published in 2017 and 2018: in the two to three years since their first publication[11] they were cited more than 350 times, once more indicating the rapid growth of and the increasing interest in the area of MLS testing.

6 Discussion

In this section, we first summarise the weaknesses of the presented approaches and we distill a set of open challenges, pointing to directions for future research.

6.1 Weaknesses

MLS testing is a new research area that has not yet defined a clear and shared research methodology. As a consequence, although existing works make important contributions to advance the state of the art, they have at the same time weaknesses in aspects that are not yet consolidated, but will definitely play an important role in the future. Such weaknesses belong to various categories, including: (1) hyper-parameter selection; (2) notions such as failure, fault, bug fixing; (3) realism and relevance of the test scenarios; (4) empirical methodology and metrics; (5) nondeterminism and statistical assessment; (6) computational cost.

Hyper-parameter selection: the proposed techniques often involve the selection of thresholds (e.g., to distinguish different levels of activation of neurons) and other hyper-parameters of the algorithms. However, such selection is often not systematic and the sensitivity of the approach to such choice is not evaluated. No guidance is provided to the reader to determine the hyper-parameter values that fit the problem at hand, and often the choice of hyper-parameters sounds quite arbitrary and is not explained in detail.

Failure, fault, bug fixing: the nature of failures and faults, as well as the process of bug fixing, change substantially or require profound adaptations when considering MLSs. However, in the literature, no consolidated terminology has been introduced yet, which affects the evaluation of effectiveness of the proposed approaches. For instance, in our surveyed papers, often a misclassification (for classifiers) or a high prediction error (for

[11] Their preprint versions were first available on May 18th, 2017 and August 28th, 2017, respectively.

regressors) are regarded as failures of the ML model. However, any ML model is by construction expected to misclassify some inputs, or to produce a nonzero prediction error on some inputs. Indeed, in real world scenarios, it is quite unrealistic to achieve perfect classification (accuracy = 1.0) or prediction scores (MSE = 0.0).

Moreover, MLS testing techniques should not solely expose misclassifications and prediction errors at the ML model level, but rather look at the side-effects of such inaccuracies at the overall system level. Individual misclassifications (or individual mis-predictions) are suboptimal definitions of failures if the whole MLS is considered, because they may have no consequences, or, on the contrary, may lead the overall system to deviate significantly from its requirements and result in a failure. For example, in the self-driving car domain, a single mis-prediction of the steering angle component may be irrelevant to the overall driving experience, if done in a situation in which the overall system is able to compensate its effects with appropriate mitigation strategies. On the other hand, there may be cases in which a single mis-prediction is responsible for a chain of subsequent mis-predictions that ultimately leads to a crash. Thus, the usage of the output produced by these approaches is sometimes unclear as the detection of misclassifications or prediction errors does not necessarily indicate that the ML model is faulty and insufficient for the task assigned to it. In other words, it is unclear if the discovered faults are real faults.

For what concerns the actions to be taken when a fault is detected, most approaches suggest re-training and show that re-training is effective in handling the adversarial/corner-case inputs that caused the misbehaviours. However, re-training is the right corrective action only if the discovered fault is associated with the training phase, but not all ML faults are necessarily due to inadequate training, as reported in existing taxonomies of deep learning faults [68, 97, 100]. For instance, the model structure (not its training) may be inadequate for the task being considered. In such a case, re-training is not the appropriate bug fixing action. Additionally, while it is quite straightforward that re-training on adversarial inputs will allow such cases to be handled properly by the re-trained system, this action may have detrimental effects on the effectiveness of the system in real operating conditions if adversarial examples are not representative of realistic cases. Thus, the response of the system to adversarial examples may become unclear or unpredictable, worsening the reliability of the MLS with respect to the original system.

Realism: Artificial adversarial inputs are regarded as test cases that expose faults, regardless of their likelihood of occurrence. Realism of generated inputs is not assessed. Transformations (e.g., image transformations) that are supposed to preserve the semantics are not checked to respect such assumption, e.g., by means of a human study. In fact, small input perturbations do not necessarily ensure that the resulting (adversarial) input is a relevant/realistic test case. For instance, if the starting point is already troublesome to classify for humans (e.g., a handwritten digit "7" that looks already like a "1"), the transformation may end up with an input that falls beyond the validity domain of the ML model [120]. Hence, a misbehaviour on such input cannot be regarded as a failure and does not point to any fault. Sometimes the test inputs used to show the effectiveness of the proposed approach are artificial and unrealistic. Correspondingly, it is quite expected that the ML model may misbehave when applied to them. The issue of realism affects also the simulated environments where the ML based system operates during testing. In fact, often over-simplified models of the environment are adopted, making the possibility to generalise the results to the real operational environment at least questionable. The same issue affects existing proposals of ML mutation operators, which do not mimic real faults and introduce unrealistic issues. For instance, mutation operators that modify the weights of a neural network after it has been trained do not match any ML fault reported in the existing taxonomies [68, 97, 100] and look quite unrealistic.

Empirical methodology and metrics: Metrics have been proposed (e.g., to prioritise test cases, to measure adequacy of test data) without any form of theoretical foundation, or empirical validation [78, 79]. Only their capability to react to adversarial examples is demonstrated, i.e., (accuracy) metrics improve if the test suite is augmented with adversarial examples. However, their usefulness in practice, to improve the thoroughness of MLS testing, is still to be demonstrated. Overall, empirical validation of MLS testing techniques lacks a shared, well-defined methodology. A small number of datasets (often just one) is used for the empirical validation. Empirical

validation focuses mostly on classification problems. Regression problems are rarely included, especially because they often require nontrivial adaptation of the proposed techniques. In general, empirical validation does not aim for the exposure of real faults, since no dataset of real faults is available yet, and surrogate evaluation targets (inaccuracies on adversarial inputs) are immature and quite unconvincing.

Nondeterminism: ML systems behave differently if trained on a different dataset or even when re-trained on the same dataset, if the training algorithm is nondeterministic (this is always the case of neural networks, whose weights are initially assigned a small random value). However, assessment of MLS testing techniques often do not consider that results may differ upon re-execution, if the system is re-trained. A notable example is the notion of mutation killing. In fact, drop of accuracy or prediction error cannot be used as the only signal of a test suite being able to kill a mutant. In fact, the exposed performance drop may be just due to the re-training of the mutated ML system, not to the effectiveness of the test suite. In fact, even the original, un-mutated system may be "killed" by a test suite if it is trained multiple times and its performance (accuracy or prediction error) drops upon some re-training.

Computational cost: Test generation techniques may require the execution of the system under test to assess the quality (e.g., adequacy) of the generated test scenarios. This may be computationally expensive, for instance, if execution involves complex and long-running simulations. The majority of existing approaches do not consider the algorithmic and runtime cost of the proposed solutions. However, practical adoption of the proposed MLS testing techniques in the daily activity of developers demands for computationally affordable solutions that can deliver their results in a reasonable time.

6.2 Open Challenges

Most of the weaknesses discussed in the previous section can be turned into open challenges and opportunities for future research. Correspondingly, many of the categories of weaknesses appear in this section as open challenges. Our analysis of the literature highlighted several opportunities for future research that can potentially lead to major breakthroughs in the area. We can group the open challenges into the following categories: (1) hyper-parameters; (2) failures, faults, bug fixing; (3) empirical methodology and metrics; (4) realism; (5) nondeterminism; (6) computational cost; (7) scope of MLS testing.

Hyper-parameters: we should define and adopt a systematic methodology for the selection and reporting of hyper-parameters of the proposed MLS testing techniques. Assessment of the sensitivity of the proposed techniques to their hyper-parameters should become standard practice.

Exposing failures, finding faults, fixing bugs: ML models should be evaluated in the context of the software system they are part of. In fact, inaccuracies of an isolated ML model (misclassifications or prediction errors) may or may not have consequences that can be regarded as failures at the system level. Hence, the effectiveness of test input generators should be measured on relevant, practical usage scenarios where a failure is a failure of the whole system, not an inaccuracy of the ML model alone. Indeed, this is what distinguishes the software engineering perspective on testing from the machine learning one. Correspondingly, bug fixing should amount to any intervention (e.g., on the model architecture or the training phase) to improve the system level performance, so as to reduce the possibility of system level failures. Hence, bug fixing may include, but should not be limited to, re-training.

Empirical methodology and metrics: We should introduce and adopt a solid empirical validation methodology, with associated metrics, that can assess the effectiveness of the proposed testing techniques directly, without resorting to proxies (e.g., ML inaccuracies, such as misclassifications or prediction errors). The goal is to be able to show that the proposed testing approaches are effective in exposing and fixing relevant ML issues, i.e., issues that can make the whole system fail in realistic execution scenarios, not just inaccuracies of the ML model alone, because the latter are intrinsic to the process of learning a behaviour from a training dataset and are hence unavoidable: the overall system should respect the functional requirements, despite the occasional occurrence of ML inaccuracies. Correspondingly, we need a solid benchmark for the evaluation of the proposed approaches. Such repositories exist in traditional software engineering research, such as SIR [85], Defects4J [103], SF100 [88], and BugsJS [92]. This has not been the case, however, for ML techniques in software engineering. Ideally, such a benchmark should include a variety of real ML based systems, not just simple classifiers, with known ML faults. It should include both systems that make use of regressors and systems that make use of classifiers. When it comes to systems that use regressors (e.g., for the prediction of the steering angle in self-driving cars), an open challenge is the definition of the oracle, which could take the form of metamorphic relations. Indeed, each application domain has its own constraints and requirements. More instances of domain specific oracles should be made available to ensure a fair assessment of the detection of system failures.

Realism: ML models produce an output for any input in their huge input domain, but the system that uses those ML models has usually a much narrower validity domain. For instance, an ML regressor that predicts the steering angle of a self-driving car from the image collected by its camera will produce a steering angle for any input image, including pictures that can never be taken by the car camera, pictures that do not represent any road and pictures that contain just noise or geometrical shapes. The frontier between valid inputs and invalid ones can be blurred and difficult to define, but MLS testing techniques should focus on the generation of inputs within the validity domain of the overall system in which the ML model is used [120]. Otherwise, the detected misbehaviours are only hypothetical and do not represent any problematic scenario that can occur in the real world. Techniques that manipulate the inputs to artificially generate new inputs (e.g., during the test generation process or in the definition of metamorphic relations) have the same problem: when an input within the validity domain of the system is mutated and becomes a new potential input, the result should be shown to be still a valid and realistic input for the system. Some of the proposed metamorphic relations (e.g., introducing black bands upon image translation) do not respect such constraints. We need to define a set of input transformations that produce realistic scenarios, not artificial ones. In the case of metamorphic relations, such transformations are also supposed to be semantic preserving. Human studies may be needed to ensure that semantic preservation is indeed guaranteed by the proposed metamorphic relations. Realism of the transformed inputs could be also assessed empirically in human studies. We should produce a catalog of input transformations that are ensured to preserve realism, a subset of which is also ensured to preserve the semantics of the input (metamorphic relations). When a misbehaviour is exposed, we should be able to distinguish between realistic/possible vs unrealistic/unlikely test scenarios (e.g., by a measure of realism/likelihood). ML mutations are also affected by the realism issue. In fact, mutations that do not match any real fault may be useless in exposing failures of the MLS. Ideally, mutation operators should be based on real faults that developers may introduce into ML components when these are designed, developed and trained. Moreover, useful mutation operators are also required to be nontrivial as well as not impossible to kill, because they are otherwise not discriminative at all [101]. Simulators are often used to run ML based systems in a controlled environment, where testing is faster and safer than in the production environment. However, the degree of realism of the simulated environment affects the capability to expose all and only relevant faults. We need complex open source simulation environments that support rich models of the execution contexts (e.g., complete models of the environment where a self-driving car operates, including pedestrians, other vehicles, buildings, trees, obstacles, signs, and traffic lights).

Nondeterminism: Effectiveness of the proposed MLS testing techniques should take the intrinsic nondeterminism of ML training into account. Observing a change of behaviour is not enough to show that the MLS testing technique is effective, because the same change may be observed when training the system multiple times on the same dataset. A positive effect of the proposed testing technique can be meaningfully reported only when the distribution of the effectiveness metric values is shown to be statistically different from the distribution observed upon multiple training. For instance, a mutant can be said to be killed by a test suite only if a statistical test, such as Wilcoxon, can reject the null hypothesis [101] (in this case, the null hypothesis states that the accuracy distributions for the mutant and for the original system are the same). In general, a statistical test should be executed to compare the distributions of the effectiveness metrics upon multiple training of the original system vs the mutated/faulty version.

Computational cost: MLS testing techniques may require computationally expensive executions of the ML system and studies often report long timeouts (e.g., days) and substantial computational resources (e.g., high performance computing clusters), which may be unavailable to the ML system testers. Further research is needed to speed up test generation techniques that require test case execution, e.g., by resorting to surrogate models that can estimate the output of an ML based system without executing it (see preliminary results by Beglerovic et al. [6]).

Scope of the MLS testing research area: MLS testing is not a well established and consolidated area yet. As such, it lacks a shared terminology, experimental methodology, agreed assessment metrics and benchmark. Its positioning in the broad discipline of computer science is also not completely clear. This was quite apparent when we collected the papers relevant for this systematic mapping. In fact, while most MLS testing papers appeared in software engineering and software testing venues, other important works (reached through snowballing) were published at machine learning, autonomous vehicles and other AI-related conferences. The tagging of the relevant papers (e.g., in arXiv) is usually "software engineering", but this is often accompanied by "machine learning" and occasionally only the latter tag is used. This situation points to the need for finding the most appropriate venue and community for research works on MLS testing. Some attempts to create new events devoted to this new field are already emerging, such as the DeepTest workshop,[12] co-located with ICSE, or the IEEE International Conference On Artificial Intelligence Testing.[13]

7 Related Work

To the best of our knowledge, eight secondary studies have been conducted in the area of MLS testing. Among them, the works by Sherin et al. [124] and J. M. Zhang et al. [133] are the most related to this mapping in terms of goals and approach. Our discussion of the former work is based on the preprint version, since it is not published in any peer-reviewed venue at the time of writing.

Similarly to our study, Sherin et al. [124] performed a systematic mapping on testing techniques for machine learning programs. They considered 37 papers in total. Sherin et al. have mostly focused on the analysis of ML models in isolation, with a few exceptions, such as the work by Tuncali et al. [55]. In contrast, our mapping is not limited to ML model testing, but also considers testing of whole systems that use one or multiple ML models. Moreover, in the work by Sherin et al. [124] the considered studies are classified based only on the type of testing approach, e.g., mutation, combinatorial, evolutionary. We instead investigate multiple aspects of the testing approaches, and provide a finer-grained, multi-dimensional classification, by considering, among the others, the used/proposed adequacy criteria, the algorithms for automated input generation, and the test oracles. There are also remarkable methodological differences between the two mappings: in the data extraction phase, Sherin

[12] `https://deeptestconf.github.io`

[13] `http://ieeeaitests.com/`

et al. [124] identified beforehand a closed list of values for each attribute, and assessors assigned a value from the list to each paper. Hence, their conclusions descend from the distribution of these attributes. Differently, we allowed the assessors to provide open answers that were later analysed, grouped and labeled, so as to report an informed discussion about addressed problems, open challenges and weaknesses of the retrieved papers. Many of them are not reported among the findings of Sherin et al. [124], such as our discussion about hyper-parameter selection, the regression problems potentially introduced when re-training is used as a corrective action, or the effect of non-deterministic training algorithms on the oracle problem. Moreover, open-ended questions allowed us to propose new concepts such as data-box access, ML model testing level, domain-specific failure and scenario coverage, which are specific for MLS testing. This allowed us to provide additional insights about weaknesses common to many existing studies, such as evaluating ML models mostly in isolation and not as a part of the whole MLS. We also identify unique open challenges not reported by Sherin et al. [124], among which: (1) evaluating whether inaccuracies of an isolated ML model have consequences that can be regarded as failures at the system level, and (2) generating inputs within the validity domain of the overall system in order to detect misbehaviours that can occur in the real world. In conclusion, our mapping focuses on whole MLSs, covers multiple dimensions, and provides open-ended analyses that cannot be found in the systematic mapping by Sherin et al. [124].

J. M. Zhang et al. [133] provide a comprehensive survey on MLS testing. The authors collected 128 relevant primary studies through multiple search methods. The work is broader in scope than our systematic mapping, since it considers testing aspects other than functional testing, such as security, data privacy, efficiency and interpretability, and their analysis is not restricted to the software engineering field. On the other hand, we investigated software engineering aspects not considered by J. M. Zhang et al., such as the testing levels and the type of access to the ML model; the kind of experimental evaluation (e.g., academic vs industrial); the types of ML models and systems; the availability of artefacts; the metrics adopted in the experiments; the baselines used for comparison (in case of comparative studies); the availability of experimental data; the software and hardware setup used in the experiments. We believe that both works provide useful and complementary information to researchers interested in MLS testing. However, it should be noted that their study is designed as a survey, whose goal is to provide a generic yet comprehensive overview on the body of literature related to MLS testing. Differently, our work is a systematic mapping that was driven by 33 carefully formulated research questions. Accordingly, we investigated in detail the retrieved papers along multiple dimensions (e.g., test level, access, metrics) to synthesise new evidence based on the analysis of the literature. For instance, analysis of the testing level allowed us to identify a gap between model level faults and system level failures, which definitely deserves further investigation. The high computational costs for the execution of test cases in simulators challenge researchers to find time-efficient approaches for test generation. As in the survey by J. M. Zhang et al., we also identify nondeterminism as a weakness of the existing MLS testing techniques. Zhang et al. address it by proposing flaky test detection as an open challenge for research. Instead, we suggest to execute statistical tests to compare the distributions of the effectiveness metrics upon multiple trainings.

In addition to the two studies discussed above, there are six secondary studies that investigate research areas partially overlapping with ML testing.

Masuda et al. [115] report a short survey about software quality of learning-based applications. They collected 101 papers from artificial intelligence and software engineering conferences, including five magazines, and tagged each paper with one keyword from an artificial intelligence vocabulary. The research questions are quite specific to quality, verification, and design aspects. Differently, our work focuses on functional aspects of MLS testing, for which we classified the retrieved papers considering multiple dimensions and a set of research questions biased toward software testing.

Braiek and Khomh [77] reviewed 39 papers about verification and testing practices for ensuring the quality of ML models and their training data. Similarly to Braiek and Khomh [77], we outline the main features of the proposed testing approaches. However, the scope of our mapping is wider, since we considered techniques that test systems containing one or more ML models, not just isolated ML models. Moreover, we performed

a deeper decomposition of the testing aspects into multiple, detailed research questions, and investigated how the considered techniques have been evaluated.

Borg et al. [76] performed a literature review focused on challenges and solution proposals for verification & validation and safety analysis (including test case design) in the automotive industry. The findings were validated through a semi-structured survey submitted to practitioners from the automotive domain. On the contrary, our scope includes testing approaches for MLSs in any domain, which includes, but it is not limited to the automotive domain.

Huang et al. [96] surveyed 178 papers on safety and trustworthiness of deep neural networks, including aspects like verification, testing, adversarial attacks, and interpretability. They reviewed testing as one of the NN certification approaches along with verification. On the contrary, we focused specifically on testing approaches and on systems that are not limited to isolated NN: any system that includes an NN as a component (or more generally an ML model) is in the scope of our mapping. Moreover, the authors focused only on coverage criteria, test case generation and model-level mutation, whereas we included a more thorough set of dimensions and research questions.

Ma et al. [113] surveyed 223 papers about quality assurance and security of DL components and proposed a DL software development process. Our work has a different scope, i.e., we focus on testing techniques instead of quality/security assurance. As for the analysis, they identified the main challenges but did not investigate the features of the considered techniques. On the other hand, we answer research questions designed to provide a detailed characterisation of several features of the proposed approaches.

Hains et al. [93] surveyed 26 papers about secure DL, including works about testing DL systems. We differ from this work as we conducted a systematic mapping instead of a survey, and we focus on functional testing rather than on safety and security assurance.

8 Conclusions and Future Work

A fast growing trend in nowadays software is to adopt ML components, which are very effective in practice, but raise novel and unprecedented challenges with respect to testing their correctness and dependability. In this paper we adopted a rigorous methodology to gain a better understanding of the trends and the state-of-the-art concerning MLS testing, along with the limitations they bear. We have identified several key open challenges that researchers should focus on in future work. From an initial pool of 1'175 papers retrieved from scientific databases, we systematically selected 47 papers and complemented them with other 23 papers obtained through snowballing. We analysed the final pool of 70 papers using a large number of research dimensions.

Our findings show that the most prevalent testing problems addressed in these papers are the oracle problem (19%), followed by the design and adoption of test adequacy criteria (17%). This is further confirmed by our analysis of test artefacts produced by each paper, where 60% of the artefacts are test inputs and 22% are test oracles. For the generation of test inputs, the most popular techniques are input mutation (31%) and search-based approaches (25%). In half of the papers, the test adequacy criterion to assess the quality of the inputs is measured using recently proposed metrics based on neuron activation of the NN. The main type of oracle used is the metamorphic oracle (44%), followed by domain-specific failures (20%). The papers mostly address model level testing (63%), with only one paper focusing on integration level testing. When it comes to the access to the system under test, white box access is required in 43% and black box in 41% of the cases.

We further investigated whether the proposed approaches and the produced experimental data are shared with the research community by the authors of each publication. Our results show that these artefacts are not available for more than 74% of papers. Moreover, a very small fraction of the papers report details about the experimental process, such as software setup (31%), hardware setup (27%) and allocated time budget (11%).

The demographic overview shows that 51 out of 70 papers were produced in recent years, i.e. starting from 2016. When it comes to venue, 40 of the papers (57%) have been published in conferences.

Overall, our analysis revealed that novel testing techniques need to be devised to account for the peculiar characteristics of MLS and highlights the most challenging problems, among which the notion of fault for an ML system, the precise characterisation of the decision boundaries of such systems, the generation of realistic inputs, within the validity domain of the MLS under test, and the creation of automated and reliable oracles for systems governed by intrinsic uncertainty, affected by nondeterminism and dealing with a huge input space.

Our suggestions for future work include the development of common frameworks and benchmarks to collect and evaluate the state-of-the-art techniques, to avoid relying on ad-hoc solutions and to support replication and comparative research. We believe that the findings of this work can help newcomers to the field in better understanding the research landscape, as well as active researchers in defining the roadmap toward future testing techniques for ML-based software.

Acknowledgements

This work was partially supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

List of Selected Primary Studies

1. Abdessalem, R.B., Nejati, S., Briand, L.C., Stifter, T.: Testing advanced driver assistance systems using multi-objective search and neural networks. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE, pp. 63–74 (2016)
2. Abdessalem, R.B., Nejati, S., Briand, L.C., Stifter, T.: Testing vision-based control systems using learnable evolutionary algorithms. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, pp. 1016–1026. ACM, New York, NY, USA (2018). DOI 10.1145/3180155.3180160. URL http://doi.acm.org/10.1145/3180155.3180160
3. Abdessalem, R.B., Panichella, A., Nejati, S., Briand, L.C., Stifter, T.: Testing autonomous cars for feature interaction failures using many-objective search. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, pp. 143–154. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238192. URL http://doi.acm.org/10.1145/3238147.3238192
4. Abeysirigoonawardena, Y., Shkurti, F., Dudek, G.: Generating adversarial driving scenarios in high-fidelity simulators. In: International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019, pp. 8271–8277 (2019). DOI 10.1109/ICRA.2019.8793740. URL https://doi.org/10.1109/ICRA.2019.8793740
5. Aniculaesei, A., Grieser, J., Rausch, A., Rehfeldt, K., Warnecke, T.: Towards a holistic software systems engineering approach for dependable autonomous systems. In: Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems - SEFAIS. ACM Press (2018). DOI 10.1145/3194085.3194091
6. Beglerovic, H., Stolz, M., Horn, M.: Testing of autonomous vehicles using surrogate models and stochastic optimization. In: 20th IEEE International Conference on Intelligent Transportation Systems, ITSC '17, pp. 1–6 (2017)
7. Bolte, J., Bar, A., Lipinski, D., Fingscheidt, T.: Towards corner case detection for autonomous driving. In: 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 438–445 (2019). DOI 10.1109/IVS.2019.8813817
8. Bühler, O., Wegener, J.: Automatic testing of an autonomous parking system using evolutionary computation. Tech. rep., SAE Technical Paper (2004)
9. Byun, T., Sharma, V., Vijayakumar, A., Rayadurgam, S., Cofer, D.: Input prioritization for testing neural networks (2019)
10. Cheng, C.H., Huang, C.H., Yasuoka, H.: Quantitative projection coverage for testing ml-enabled autonomous systems. In: S.K. Lahiri, C. Wang (eds.) Automated Technology for Verification and Analysis, pp. 126–142. Springer International Publishing, Cham (2018)
11. Cheng, D., Cao, C., Xu, C., Ma, X.: Manifesting bugs in machine learning code: An explorative study with mutation testing. In: 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 313–324. IEEE (2018)
12. Ding, J., Kang, X., Hu, X.: Validating a deep learning framework by metamorphic testing. In: 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), pp. 28–34 (2017). DOI 10.1109/MET.2017.2
13. Du, X., Xie, X., Li, Y., Ma, L., Liu, Y., Zhao, J.: Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 477–487. ACM (2019)
14. Dwarakanath, A., Ahuja, M., Sikand, S., Rao, R.M., Bose, R.P.J.C., Dubash, N., Podder, S.: Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, pp. 118–128. ACM, New York, NY, USA (2018). DOI 10.1145/3213846.3213858. URL http://doi.acm.org/10.1145/3213846.3213858

15. Eniser, H.F., Gerasimou, S., Sen, A.: Deepfault: Fault localization for deep neural networks. In: R. Hähnle, W. van der Aalst (eds.) Fundamental Approaches to Software Engineering, pp. 171–191. Springer International Publishing, Cham (2019)
16. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: A language for scenario specification and scene generation. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, pp. 63–78. ACM, New York, NY, USA (2019). DOI 10.1145/3314221.3314633. URL http://doi.acm.org/10.1145/3314221.3314633
17. Gopinath, D., Wang, K., Zhang, M., Pasareanu, C.S., Khurshid, S.: Symbolic execution for deep neural networks. CoRR **abs/1807.10439** (2018). URL http://arxiv.org/abs/1807.10439
18. Groce, A., Kulesza, T., Zhang, C., Shamasunder, S., Burnett, M., Wong, W., Stumpf, S., Das, S., Shinsel, A., Bice, F., McIntosh, K.: You are the only possible oracle: Effective test selection for end users of interactive machine learning systems. IEEE Transactions on Software Engineering **40**(3), 307–323 (2014). DOI 10.1109/TSE.2013.59
19. Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J.: Dlfuzz: differential fuzzing testing of deep learning systems. In: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE, pp. 739–743 (2018)
20. Henriksson, J., Berger, C., Borg, M., Tornberg, L., Englund, C., Sathyamoorthy, S.R., Ursing, S.: Towards structured evaluation of deep neural network supervisors. In: 2019 IEEE International Conference On Artificial Intelligence Testing (AITest). IEEE (2019). DOI 10.1109/aitest.2019.00-12
21. Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy. In: Proceedings of the 41st International Conference on Software Engineering, ICSE, pp. 1039–1049 (2019)
22. Klueck, F., Li, Y., Nica, M., Tao, J., Wotawa, F.: Using ontologies for test suites generation for automated and autonomous driving functions. In: 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 118–123 (2018). DOI 10.1109/ISSREW.2018.00-20
23. Li, G., Pattabiraman, K., DeBardeleben, N.: Tensorfi: A configurable fault injector for tensorflow applications. In: 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 313–320 (2018). DOI 10.1109/ISSREW.2018.00024
24. Li, L., Huang, W., Liu, Y., Zheng, N., Wang, F.: Intelligence testing for autonomous vehicles: A new approach. IEEE Transactions on Intelligent Vehicles **1**(2), 158–166 (2016). DOI 10.1109/TIV.2016.2608003
25. Ma, L., Juefei-Xu, F., Xue, M., Li, B., Li, L., Liu, Y., Zhao, J.: Deepct: Tomographic combinatorial testing for deep learning systems. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 614–618. IEEE (2019)
26. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., Wang, Y.: Deepgauge: Multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, pp. 120–131. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238202. URL http://doi.acm.org/10.1145/3238147.3238202
27. Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., et al.: Deepmutation: Mutation testing of deep learning systems. In: 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), pp. 100–111. IEEE (2018)
28. Ma, S., Liu, Y., Lee, W.C., Zhang, X., Grama, A.: Mode: Automated neural network model debugging via state differential analysis and input selection. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, pp. 175–186. ACM, New York, NY, USA (2018). DOI 10.1145/3236024.3236082. URL http://doi.acm.org/10.1145/3236024.3236082
29. Majumdar, R., Mathur, A., Pirron, M., Stegner, L., Zufferey, D.: Paracosm: A language and tool for testing autonomous driving systems
30. Mullins, G., Stankiewicz, P., Hawthorne, R., Gupta, S.: Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. Journal of Systems and Software **137**, 197–215 (2018). DOI 10.1016/j.jss.2017.10.031. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85037521968
31. Murphy, C., Kaiser, G., Arias, M.: An approach to software testing of machine learning applications. SEKE 2007 p. 167 (2007)
32. Murphy, C., Kaiser, G., Arias, M.: Parameterizing random test data according to equivalence classes. In: Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), pp. 38–41. ACM (2007)
33. Murphy, C., Kaiser, G., Hu, L., Wu, L.: Properties of machine learning applications for use in metamorphic testing. SEKE 2008 p. 867 (2008)
34. Murphy, C., Kaiser, G.E.: Improving the dependability of machine learning applications (2008)
35. Murphy, C., Shen, K., Kaiser, G.: Automatic system testing of programs without test oracles. In: Proceedings of the eighteenth international symposium on Software testing and analysis, pp. 189–200. ACM (2009)
36. Nakajima, S.: Generalized oracle for testing machine learning computer programs. In: A. Cerone, M. Roveri (eds.) Software Engineering and Formal Methods, pp. 174–179. Springer International Publishing, Cham (2018)
37. Nakajima, S.: Dataset diversity for metamorphic testing of machine learning software. In: Structured Object-Oriented Formal Language and Method, pp. 21–38. Springer International Publishing (2019)

38. Nakajima, S., Bui, H.N.: Dataset coverage for testing machine learning computer programs. In: Proceedings of the 23rd Asia-Pacific Software Engineering Conference, APSEC '16, pp. 297–304 (2016). DOI 10.1109/APSEC.2016.049
39. Odena, A., Olsson, C., Andersen, D., Goodfellow, I.: TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In: K. Chaudhuri, R. Salakhutdinov (eds.) Proceedings of the 36th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 97, pp. 4901–4911. PMLR, Long Beach, California, USA (2019). URL `http://proceedings.mlr.press/v97/odena19a.html`
40. de Oliveira Neves, V., Delamaro, M.E., Masiero, P.C.: Combination and mutation strategies to support test data generation in the context of autonomous vehicles. IJES **8**(5/6), 464–482 (2016). DOI 10.1504/IJES.2016.10001345. URL `https://doi.org/10.1504/IJES.2016.10001345`
41. Patel, N., Saridena, A.N., Choromanska, A., Krishnamurthy, P., Khorrami, F.: Adversarial learning-based on-line anomaly monitoring for assured autonomy. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018, pp. 6149–6154 (2018). DOI 10.1109/IROS.2018.8593375. URL `https://doi.org/10.1109/IROS.2018.8593375`
42. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 1–18 (2017)
43. Qin, Y., Wang, H., Xu, C., Ma, X., Lu, J.: Syneva: Evaluating ml programs by mirror program synthesis. In: 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 171–182. IEEE (2018)
44. Rubaiyat, A.H.M., Qin, Y., Alemzadeh, H.: Experimental resilience assessment of an open-source driving agent. In: 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 54–63 (2018). DOI 10.1109/PRDC.2018.00016
45. Saha, P., Kanewala, U.: Fault detection effectiveness of metamorphic relations developed for testing supervised classifiers. In: 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), pp. 157–164 (2019). DOI 10.1109/AITest.2019.00019
46. Sekhon, J., Fleming, C.: Towards improved testing for deep learning. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 85–88 (2019). DOI 10.1109/ICSE-NIER.2019.00030
47. Shen, W., Wan, J., Chen, Z.: Munn: Mutation analysis of neural networks. In: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 108–115. IEEE (2018)
48. Shi, Q., Wan, J., Feng, Y., Fang, C., Chen, Z.: Deepgini: Prioritizing massive tests to reduce labeling cost (2019)
49. Spieker, H., Gotlieb, A.: Towards testing of deep learning systems with training set reduction. CoRR **abs/1901.04169** (2019). URL `http://arxiv.org/abs/1901.04169`
50. Strickland, M., Fainekos, G., Ben Amor, H.: Deep predictive models for collision risk assessment in autonomous driving. In: 2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Proceedings - IEEE International Conference on Robotics and Automation, pp. 4685–4692. Institute of Electrical and Electronics Engineers Inc. (2018). DOI 10.1109/ICRA.2018.8461160
51. Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R.: Testing deep neural networks (2018)
52. Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D.: Concolic testing for deep neural networks. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE '18, pp. 109–119. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238172. URL `http://doi.acm.org/10.1145/3238147.3238172`
53. Tian, Y., Pei, K., Jana, S., Ray, B.: Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, pp. 303–314. ACM, New York, NY, USA (2018). DOI 10.1145/3180155.3180220. URL `http://doi.acm.org/10.1145/3180155.3180220`
54. Tuncali, C.E., Fainekos, G.: Rapidly-exploring random trees-based test generation for autonomous vehicles. CoRR **abs/1903.10629** (2019). URL `http://arxiv.org/abs/1903.10629`
55. Tuncali, C.E., Fainekos, G., Ito, H., Kapinski, J.: Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 1555–1562. IEEE (2018)
56. Udeshi, S., Arora, P., Chattopadhyay, S.: Automated directed fairness testing. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, pp. 98–108. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238165. URL `http://doi.acm.org/10.1145/3238147.3238165`
57. Udeshi, S., Chattopadhyay, S.: Grammar based directed testing of machine learning systems (2019)
58. Uesato, J., Kumar, A., Szepesvári, C., Erez, T., Ruderman, A., Anderson, K., Dvijotham, K.D., Heess, N., Kohli, P.: Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019 (2019)
59. Wang, J., Dong, G., Sun, J., Wang, X., Zhang, P.: Adversarial sample detection for deep neural network through model mutation testing. In: Proceedings of the 41st International Conference on Software Engineering, ICSE '19, pp. 1245–1256. IEEE Press, Piscataway, NJ, USA (2019). DOI 10.1109/ICSE.2019.00126. URL `https://doi.org/10.1109/ICSE.2019.00126`
60. Wolschke, C., Kuhn, T., Rombach, D., Liggesmeyer, P.: Observation based creation of minimal test suites for autonomous vehicles. In: 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 294–301 (2017). DOI 10.1109/ISSREW.2017.46

61. Wolschke, C., Rombach, D., Liggesmeyer, P., Kuhn, T.: Mining test inputs for autonomous vehicles. In: K. Berns, K. Dressler, P. Fleischmann, D. Görges, R. Kalmar, B. Sauer, N. Stephan, R. Teutsch, M. Thul (eds.) Commercial Vehicle Technology 2018, pp. 102–113. Springer Fachmedien Wiesbaden, Wiesbaden (2018)
62. Xie, X., Ho, J.W., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y.: Testing and validating machine learning classifiers by metamorphic testing. Journal of Systems and Software **84**(4), 544–558 (2011)
63. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA '19, pp. 146–157. ACM, New York, NY, USA (2019). DOI 10.1145/3293882.3330579. URL http://doi.acm.org/10.1145/3293882.3330579
64. Xie, X., Zhang, Z., Yueh Chen, T., Liu, Y., Poon, P.L., Xu, B.: Mettle: A metamorphic testing approach to validating unsupervised machine learning methods. arXiv preprint arXiv:1807.10453 (2018)
65. Zhang, J., Jing, X., Zhang, W., Wang, H., Dong, Y.: Improve the quality of arc systems based on the metamorphic testing. In: 2016 International Symposium on System and Software Reliability (ISSSR), pp. 137–141 (2016). DOI 10.1109/ISSSR.2016.029
66. Zhang, L., Sun, X., Li, Y., Zhang, Z.: A noise-sensitivity-analysis-based test prioritization technique for deep neural networks. CoRR **abs/1901.00054** (2019). URL http://arxiv.org/abs/1901.00054
67. Zhang, M., Zhang, Y., Zhang, L., Liu, C., Khurshid, S.: Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE, pp. 132–142 (2018)
68. Zhang, Y., Chen, Y., Cheung, S.C., Xiong, Y., Zhang, L.: An empirical study on tensorflow program bugs. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, pp. 129–140. ACM, New York, NY, USA (2018). DOI 10.1145/3213846.3213866. URL http://doi.acm.org/10.1145/3213846.3213866
69. Zhao, X., Gao, X.: An ai software test method based on scene deductive approach. In: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 14–20. IEEE (2018)
70. Zheng, W., Wang, W., Liu, D., Zhang, C., Zeng, Q., Deng, Y., Yang, W., He, P., Xie, T.: Testing untestable neural machine translation: an industrial case. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, pp. 314–315. IEEE Press (2019)

References

71. Ali, N.B., Petersen, K.: Evaluating strategies for study selection in systematic literature studies. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, pp. 45:1–45:4. ACM, New York, NY, USA (2014). DOI 10.1145/2652524.2652557. URL http://doi.acm.org/10.1145/2652524.2652557
72. Altman, N.S.: An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician **46**(3), 175–185 (1992). DOI 10.1080/00031305.1992.10475879
73. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: Effective and explainable detection of android malware in your pocket. In: Ndss, vol. 14, pp. 23–26 (2014)
74. Arthur, D., Vassilvitskii, S.: K-means++: The advantages of careful seeding. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, pp. 1027–1035. Society for Industrial and Applied Mathematics (2007). URL http://dl.acm.org/citation.cfm?id=1283383.1283494
75. Bishop, C.M.: Pattern recognition and machine learning. Springer Science+ Business Media (2006)
76. Borg, M., Englund, C., Wnuk, K., Duran, B., Levandowski, C., Gao, S., Tan, Y., Kaijser, H., Lönn, H., Törnqvist, J.: Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. Journal of Automotive Software Engineering **1**, 1–19 (2019). DOI https://doi.org/10.2991/jase.d.190131.001. URL https://doi.org/10.2991/jase.d.190131.001
77. Braiek, H.B., Khomh, F.: On testing machine learning programs (2018)
78. Briand, L.C., Daly, J.W., Wüst, J.: A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering **3**(1), 65–117 (1998)
79. Briand, L.C., Daly, J.W., Wüst, J.: A unified framework for coupling measurement in object-oriented systems. IEEE Trans. Software Eng. **25**(1), 91–121 (1999)
80. Campos, G.O., Zimek, A., Sander, J., Campello, R.J., Micenková, B., Schubert, E., Assent, I., Houle, M.E.: On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. Data Min. Knowl. Discov. **30**(4), 891–927 (2016). DOI 10.1007/s10618-015-0444-8. URL http://dx.doi.org/10.1007/s10618-015-0444-8
81. Cerf, V.G.: A comprehensive self-driving car test. Commun. ACM **61**(2), 7–7 (2018). DOI 10.1145/3177753. URL http://doi.acm.org/10.1145/3177753
82. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3213–3223 (2016)

83. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995). DOI 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411
84. Davis, M.D., Weyuker, E.J.: Pseudo-oracles for non-testable programs. In: Proceedings of the ACM'81 Conference, pp. 254–257. ACM (1981)
85. Do, H., Elbaum, S., Rothermel, G.: Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. EMSE **10**(4), 405–435 (2005)
86. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: S. Levine, V. Vanhoucke, K. Goldberg (eds.) Proceedings of the 1st Annual Conference on Robot Learning, *Proceedings of Machine Learning Research*, vol. 78, pp. 1–16. PMLR (2017). URL http://proceedings.mlr.press/v78/dosovitskiy17a.html
87. Dua, D., Graff, C.: UCI machine learning repository (2017). URL http://archive.ics.uci.edu/ml
88. Fraser, G., Arcuri, A.: Sound empirical evidence in software testing. In: Proc. of ICSE '12, pp. 178–188 (2012)
89. Gal, Y.: Uncertainty in deep learning. Ph.D. thesis, PhD thesis, University of Cambridge (2016)
90. Garousi, V., Felderer, M., Mäntylä, M.V., Rainer, A.: Benefitting from the grey literature in software engineering research (2019)
91. Gross, P., Boulanger, A., Arias, M., Waltz, D., Long, P.M., Lawson, C., Anderson, R., Koenig, M., Mastrocinque, M., Fairechio, W., Johnson, J.A., Lee, S., Doherty, F., Kressner, A.: Predicting electricity distribution feeder failures using machine learning susceptibility analysis. In: Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence - Volume 2, IAAI'06, pp. 1705–1711. AAAI Press (2006). URL http://dl.acm.org/citation.cfm?id=1597122.1597127
92. Gyimesi, P., Vancsics, B., Stocco, A., Mazinanian, D., Beszedes, A., Ferenc, R., Mesbah, A.: BugsJS: a benchmark of JavaScript bugs. In: Proceedings of the International Conference on Software Testing, Verification, and Validation (ICST). IEEE Computer Society (2019)
93. Hains, G., Jakobsson, A., Khmelevsky, Y.: Towards formal methods and software engineering for deep learning: Security, safety and productivity for dl systems development. In: 2018 Annual IEEE International Systems Conference (SysCon), pp. 1–5 (2018). DOI 10.1109/SYSCON.2018.8369576
94. Hastie, T., Tibshirani, R., Friedman, J.: The elements of statistical learning: data mining, inference and prediction, 2 edn. Springer (2009). URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/
95. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016)
96. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks (2018)
97. Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P.: Taxonomy of real faults in deep learning systems. In: Proceedings of 42nd International Conference on Software Engineering, ICSE '20, p. 12 pages. ACM (2020)
98. IEEE: Ieee standard glossary of software engineering terminology. IEEE Std 610.12-1990 **Reaffirmed 12-9-2002**, 1–84 (1990). DOI 10.1109/IEEESTD.1990.101064
99. International, T.: Prescan simulation of adas and active safety (2017). URL https://www.tassinternational.com/prescan
100. Islam, M.J., Nguyen, G., Pan, R., Rajan, H.: A comprehensive study on deep learning bug characteristics. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019, pp. 510–520. ACM, New York, NY, USA (2019). DOI 10.1145/3338906.3338955. URL http://doi.acm.org/10.1145/3338906.3338955
101. Jahangirova, G., Tonella, P.: An empirical evaluation of mutation operators for deep learning systems. In: Proceedings of the International Conference on Software Testing, Verification, and Validation (ICST). IEEE Computer Society (2020)
102. James, G., Witten, D., Hastie, T., Tibshirani, R.: An introduction to statistical learning, vol. 112. Springer (2013)
103. Just, R., Jalali, D., Ernst, M.D.: Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In: Proc. of the ISSTA '14, pp. 437–440 (2014)
104. Kaufman, L., Rousseeuw, P.: Clustering by means of medoids. Statistical Data Analysis Based on the L1-Norm and Related Methods pp. North–Holland (1987)
105. Kaufman, L., Rousseeuw, P.J.: Finding groups in data: an introduction to cluster analysis. Wiley series in Probability and Mathematical Statistics. Wiley (1990). A Wiley-Interscience publication.
106. Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering – a systematic literature review. Information and Software Technology **51**(1), 7–15 (2009). DOI https://doi.org/10.1016/j.infsof.2008.09.009. URL http://www.sciencedirect.com/science/article/pii/S0950584908001390. Special Section - Most Cited Articles in 2002 and Regular Research Papers
107. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (2007)
108. Kitchenham, B.A., Mendes, E., Travassos, G.H.: Cross versus within-company cost estimation studies: A systematic review. IEEE Transactions on Software Engineering **33**(5), 316–329 (2007). DOI 10.1109/TSE.2007.1001
109. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
110. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
111. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010). URL http://yann.lecun.com/exdb/mnist/

112. Long, P.M., Servedio, R.A.: Martingale boosting. In: P. Auer, R. Meir (eds.) Learning Theory, pp. 79–94. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
113. Ma, L., Juefei-Xu, F., Xue, M., Hu, Q., Chen, S., Li, B., Liu, Y., Zhao, J., Yin, J., See, S.: Secure deep learning engineering: A software quality assurance perspective (2018)
114. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
115. Masuda, S., Ono, K., Yasue, T., Hosokawa, N.: A survey of software quality for machine learning applications. In: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 279–284 (2018). DOI 10.1109/ICSTW.2018.00061
116. McLachlan, G.J., Basford, K.E.: Mixture models: Inference and applications to clustering, vol. 38. M. Dekker New York (1988)
117. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Ease, vol. 8, pp. 68–77 (2008)
118. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology **64**, 1–18 (2015). DOI https://doi.org/10.1016/j.infsof.2015.03.007. URL `http://www.sciencedirect.com/science/article/pii/S0950584915000646`
119. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Replication package (2019). URL `https://github.com/testingautomated/deepthoughts`
120. Riccio, V., Tonella, P.: Model-based exploration of the frontier of behaviours for deep learning system testing. In: Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM (2020)
121. Rokach, L., Maimon, O.: Data Mining With Decision Trees: Theory and Applications, 2nd edn. World Scientific Publishing Co., Inc., River Edge, NJ, USA (2014)
122. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics **20**, 53–65 (1987)
123. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural networks **61**, 85–117 (2015)
124. Sherin, S., khan, M.U., Iqbal, M.Z.: A systematic mapping study on testing of machine learning programs (2019)
125. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014)
126. Speed, T.: Statistical models: Theory and practice, revised edition by david a. freedman. International Statistical Review **78**(3), 457–458 (2010). DOI 10.1111/j.1751-5823.2010.00122_11.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-5823.2010.00122_11.x`
127. Stocco, A., Weiss, M., Calzana, M., Tonella, P.: Misbehaviour prediction for autonomous driving systems. In: Proceedings of 42nd International Conference on Software Engineering, ICSE '20, p. 12 pages. ACM (2020)
128. Stuart, R., Peter, N.: Artificial intelligence - a modern approach 3rd ed. Berkeley (2016)
129. Turing, A.M.: Computing machinery and intelligence. In: Parsing the Turing Test, pp. 23–65. Springer (2009)
130. Wieringa, R., Maiden, N., Mead, N., Rolland, C.: Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requirements engineering **11**(1), 102–107 (2006)
131. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of EASE '14, pp. 1–10 (2014)
132. Young, M., Pezzè, M.: Software Testing and Analysis: Process, Principles and Techniques. John Wiley & Sons, Inc., USA (2005)
133. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering pp. 1–1 (2020)