HAL open science

A testing approach for dependable Machine Learning systems

Cyril Cappi, Camille Chapdelaine, Laurent Gardes, Eric Jenn, Baptiste Lefevre, Sylvaine Picard, Thomas Soumarmon

To cite this version:

Cyril Cappi, Camille Chapdelaine, Laurent Gardes, Eric Jenn, Baptiste Lefevre, et al.. A testing approach for dependable Machine Learning systems. 11th European Congress on Embedded Real Time Systems (ERTS), Jun 2022, Toulouse, France. hal-03694805

HAL Id: hal-03694805

https://hal.science/hal-03694805v1

Submitted on 14 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A testing approach for dependable Machine Learning systems

Cyril Cappi(1), Camille Chapdelaine(2,3), Laurent Gardes(1), Eric Jenn(3,4), Baptiste Lefevre(4), Sylvaine Picard(2), Thomas Soumarmon(3,5)

(1) SNCF, Saint-Denis, (2) Safran Tech, Magny-les-Hameaux, (3) IRT Saint-Exupéry, Toulouse, (4) THALES AVS, Toulouse and Mérignac, (5) Continental, Toulouse

***Abstract* — In order to be used into a critical system, a software or an hardware component must come with strong evidences that the designer's intents have been correctly captured and implemented. This activity is already complex and expensive for classical systems despite a very large corpus of verification methods and tools. But it is even more complicated for systems embedding Machine Learning (ML) algorithms due to the very nature of the functions being implemented using ML and the very nature of the ML algorithms. This paper focuses on one specific verification technique, testing, for which we propose a four-pronged approach combining performance testing, robustness testing, worst-case testing, and bias testing.**

Keywords— machine learning, testing, performance, robustness, worst-case, bias, safety

I. Motivation and objectives

How to engineer mission- or safety-critical systems embedding Machine-Learning (ML) is a very hot topic raising many challenges, in particular concerning verification, validation, and certification activities [1] [2] [3]. Formal verification techniques are making their way in the domain of Artificial Intelligence (AI) and ML [4], but their applicability is still limited to specific use cases and specific properties (e.g., properties around a specific input point). As of today, verification of ML system essentially relies on the only verification technique applicable on a large scale: testing. Hence, to reach the required dependability level, a rigorous testing strategy is mandatory [5]. This is particularly true for perception systems (e.g. computer vision, natural language recognition) where the dimension of the input space is huge, making the efficiency of testing highly arguable if no appropriate strategy is defined. Testing ML implies to revisit classical components of testing, such as the definition of equivalence classes or the definition of test oracles.

From these considerations, the goals of this paper are the followings:

- propose a testing strategy for systems of perception based on ML
- identify the different categories of tests supporting this strategy
- consider how test results can be used to improve the performances of the systems.

Towards those goals, we first briefly describe the target system chosen to illustrate our approach, which is a railway signal detection system based on image recognition (Section II). Then, we introduce the generic elements that constitute a test and detail our approach (Section III). In the following sections, we successively consider four categories of tests: *performance testing* (Section IV), *robustness testing* (Section V), *worst-case testing* (Section VI), and *bias testing* (Section VII). Finally, we review related works in section VIII and conclude our work.

II. Use Case

To illustrate our approach, we consider a railway Automatic Signal Processing system (ASP). The ASP is aimed at recognizing the state of a light signal applicable to a train, a task that is currently performed by train drivers. The ASP shall remain vision-based in order to limit the impact on the infrastructure and limit the cost of its deployment.

Figure 1 gives an overview of the actions performed by a train driver. The ASP must locate the signal, check the integrity of the signaling device (e.g., it is neither broken nor maliciously modified) and determine the indication of the signal automatically. As the ASP performs a critical function with potential safety effects, it shall be certified at SIL4 level according to EN50126 and EN50128 standards. In terms of error rate, and according to studies carried out at SNCF on train drivers, a maximum error rate of 10^{-5} per signal is a sensible objective. For operational reasons, the ASP shall operate in environmental conditions and tracks contexts that are both very complex and variable. In addition, the signal can be occluded or damaged. In our case, the ASP uses ML-based algorithm.

This use case shows two interesting properties with respect to the objective of our study: it implements a simple task (recognizing a light signal) and it operates in an open and weakly structured environment.

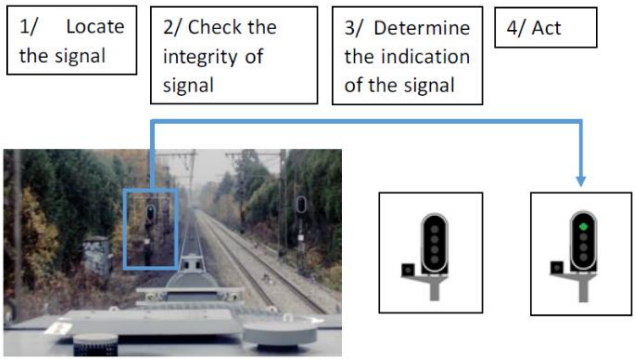

Figure 1 : Action performed by a train driver.

III. Approach

A. Context and hypotheses

In this study, we consider the following hypotheses.

- About the system:
  - The ML system is a neural network, even if the proposed approach may be applied to other ML models

- The learning phase is completed (offline learning)
- The architecture of the network is known
- The parameters of the model are known and are accessible.

- About the test objectives:
  - Intentional faults (cybersecurity) are not considered, even if the testing strategies proposed in this may also be applied in that case
  - Implementation faults are not considered. We estimate that those faults can be addressed using classical software testing techniques.

B. *What is testing*

Testing aims at demonstrating empirically that a system satisfies some properties.

A test may be used to reveal the presence of a fault in the specification or the design of a system. In that case, the test aims at *activating* some dormant fault and at *propagating* the resulting latent error up to the interface of the system to make it observable. In that case, the test may be targeted towards a specific class of faults. In the hardware domain, for instance, some tests targets stuck-at faults while, in the software domain, tests target incorrect coding of conditions or incorrect handling of input domains boundaries.

A test may also be used to verify the compliance of the system with some performance requirements. Performance may be functional (e.g., the accuracy and precision of a decision, a response time) or non-functional (e.g., the tolerance to some unintentional or intentional fault).

More generally, test is used to verify the satisfaction of some high-level properties, such as the *robustness* with respect to incorrect inputs, the *fairness* of the decisions, the *quality* of an explanation, etc.

C. *Test construction*

To perform testing, several elements are required: a *system to be tested* (the system under test), an *environment* to provide the inputs to the system, and an *oracle* to determine whether the system behaves correctly or not (i.e., the test passes or fails). In addition, a test-stopping criterion is usually defined in order to stop the testing activity since, except for trivial systems, the number of possible tests is usually infinite. The criterion may be structural (e.g., to cover all requirements, all lines of source code, all execution paths, etc.), statistical, or simply driven by the amount of effort deemed acceptable to perform this activity.

The quality of the test may also be evaluated. Referring to the previous definition of testing, the quality may be measured by the capability of the test to reveal errors. Traditionally, several strategies are used: fault/error injection to check if the test reveals the error, coverage analysis.

D. *Operational Design Domain*

The input domain depends on the purpose of the test. For performance evaluation purposes, the input domain is the "Operational Design Domain[1]" (ODD). In the automotive domain, the ODD is defined as *the specific conditions under which a given driving automation system or feature thereof is designed to function, including, but not limited to, driving modes*. In the aeronautical domain, the ODD concept is related to the concept of *foreseeable conditions*, i.e., the environment in which a system is assumed to operate, given its intended function, including operating in normal, non-normal, and emergency conditions.

Defining *precisely the ODD* is extremely difficult since it shall ideally include all the elements that may affect the function to be performed via its inputs, and all configurations (or states) of these elements. In general, the definition of the ODD cannot be formally "complete", because the environment may be too complex to characterize (i.e., it involves too many variables), or because it is simply unpredictable. Therefore, some of the operational situations remain *unknown*, and possibly unsafe [4]. As proposed in [6], the elaboration of the ODD may combine the points of view of the various actors involved in the operation and design of the system, e.g., the train driver, the image processing chain designer, etc.

Considering our use case, the ODD encompasses the state of the sensors, train, rails, signals, and, more generally, of the complete environment of the system.

Tests exercise the system on situations sampled according to the ODD. Therefore, missing a complete definition of the ODD makes testing a challenge since situation not captured by the ODD will not be considered. The next section elaborates on this challenge.

E. *Why is testing ML components difficult?*

First, it is important to notice that testing are the primary means to verify ML components today. For systems developed using non-ML techniques, testing may be replaced or complemented by other means such as formal techniques (model checking, abstract interpretation, formal proof, etc.). But even though some successes have already been obtained [7], those techniques are still in infancy in the ML domain.

ML components are particularly difficult to test for various reasons:

- The input space is often extremely large (for example all the train signal of France in all weather conditions), which poses the problem for the stopping criteria definition.
- ML components often address problems which specification is difficult to express in a comprehensive way.
- ML components often address problems where the environment is very complex and difficult to predict (see section on ODD).
- The test oracle is often a human, because the tasks performed with ML usually cannot be performed with classical methods.
- Fault models are unknown (yet) which imposes empirical performance and robustness evaluations.
- Test coverage metrics used for non-ML software are not useful since the behavior of the ML component depends essentially on data, not on control.

Test equivalence classes are difficult to define for ML components. A test equivalence is such that any test case taken in a class reveals the same faults as any other test in the same class. It is a fundamental rationale in software testing (see e.g. [8]).

[1] Defined in Section 3.17 of « Taxonomy and definitions for terms related to driving automation systems for on road motor vehicle. SAE recommanded practices J3016, Sept 2016 , https://www.sae.org/standards/j3016_201609/

F. Our strategy

To cope with the ML based system particularities, we propose a strategy based on 4 testing activities:

- **Performance testing** aims at verifying the performance of the ML model against its specification.
- **Robustness testing** aims at verifying the behavior of the system in the presence of invalid inputs or stressful environmental conditions" [9]. Robustness analysis can be seen as the system behavior analysis regarding any environmental or operating perturbation (known and unknown). The goal is different from performance testing in that robustness testing uses specifically *perturbed* inputs in or out the operational domain.
- **Worst-case testing** consists in exercising the system in the "worst" situations of the ODD and observe its performance.
- **Bias testing** aims at detecting that no bias is present in the decisions of the model itself, i.e. that the model has used relevant features to output the decisions which have led to the recorded performances.

All these activities bring complementary point of views on the component/system at hand. Figure 2 summarizes the testing approach proposed in this paper.

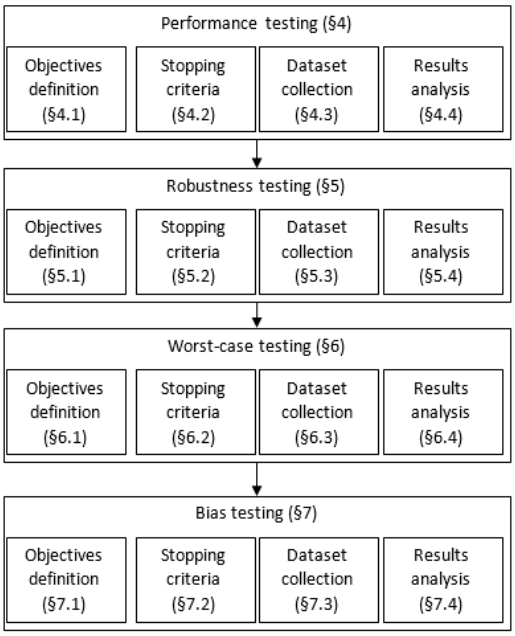

Figure 2 : Testing strategy.

G. Expected Test Results Definition

In order to check whether a test is successful or not, some pass-fail property must be expressed. The pass-fail property takes the form of a predicate involving the output produced by the system and, possibly, some reference value (expected test result, oracle, ground truth ...).

The pass-fail property normally derives from the requirements specification. The test passes if all requirements are satisfied. Unfortunately, requirements are sometimes very difficult to establish for the kind of systems concerned by ML, and so are the tests. This is why a particular care must be taken in the tests and the test data definition and creation.

The oracle may be another system possibly implemented using non-ML technologies (for testing, performance may be less important than for operations[2]) by using a back-to-back configuration or, more generally, any scheme that can be used for runtime monitoring (OOD detection, explicability-based monitoring, continuity/consistency of outputs, etc.). Here, any triggering of the monitoring is considered as a test fail condition.

A specific case is the one where the oracle is the human operator himself/herself. This applies in the situation where the system under test is embedded in an operational system (e.g., a train) and its outputs are compared with the ones of the operator. In our use case, we would for instance compare the decisions taken by the train driver with the outputs of the system under test, and check the compatibility of the decisions with the signal state reported by the system under test.

Another particular case is simulation. Here, the construction of the scenarios can be performed with respect to precise specification, and the expected output may be known a priori. Unfortunately, simulation and reality are different, and the effect of this difference are difficult to estimate. Therefore, testing based on simulation only cannot be considered as an acceptable means of verification.

Finally, some verification can still be carried out without any oracle by leveraging some invariants of the system. This is what is exploited by *metamorphic testing* techniques. See e.g., [10] [11].

H. Test Dataset quality and bias in the Dataset

As discussed in the paragraph "Test Construction", the quality of the actual tests must be verified. As we will see in the next sections, testing strongly relies on datasets. Then test validity is related to test datasets quality. Datasets quality represent different aspects like a proper dataset size, data accuracy and a small amount of untended bias.

The notion of bias in the test dataset is closely linked with the notion of representativeness. If the test dataset is biased, i.e. if some features combinations are not representative of the actual operational distribution, then the estimated performance will not hold for the general operational use. For example, if the system is only tested by day and never by night while it will be operated by night, the actual performance may significantly vary. Section IV.C proposes some strategies to ensure representativeness of the test dataset.

Biases have been identified even in well-known datasets for ML [12], and therefore, their representativeness has to be questioned [13]. The work of [12] could be used as an example to detect biases in the test dataset. Alternatively, weights for the data can be computed through an optimization problem [14]. Another way to reduce the bias is to perform a synthetic data augmentation [15].

IV. Performance Testing

A. Test objectives definition

The purpose of performance testing is to verify that the ML model meets some *performance requirement* expressed using some *performance metrics*.

1) Performance metrics

For classification problems, four basic performance metrics are usually considered [16].

[2] This may not be the case if a huge number of tests are required. See section on performance testing.

- **Accuracy:** $a = \frac{\text{number of inputs assigned to their correct class}}{\text{total number of inputs}}$
- **Precision:** $p = \frac{\text{number of inputs correctly assigned to class } i}{\text{number of inputs assigned to class } i}$
- **Recall:** $r = \frac{\text{number of inputs correctly assigned to class } i}{\text{number of inputs belonging to class } i}$
- **F1-score:** $f = 2pr \,/\, (p + r)$

These performance metrics allow measuring a mean performance on a given test dataset.

Obviously, performance metrics are application dependent. For example in the case of depth prediction from an RGB image by a Convolutionnal Neural Network, the performance metric is Root Mean Square Error (RMSE) on the depth predictions (with respect to a ground truth) [17]. In the following we consider only performance metrics related to a classification task.

2) Performance requirements

To specify performance requirements, the following information are necessary

- A **scope,** i.e., the conditions in which the performance objectives must be satisfied (ex: all operating conditions, low visibility operations, red lights, green lights…)
- One or several **performance metrics** (ex: accuracy, precision, recall…), as described in §1)
- One or several **performance objectives** expressed using the previous metrics (ex:$p \geq$ 99.9%, $a \geq$ 99.99% …)
- A **confidence level**, possibly expressed as the probability for the *actual* performance of the ML component to be greater or equal to the *estimated* performance (e.g. 99,9%, 99,99%).

The performance objectives may be class-dependent. For example, in our use case, classifying red lights correctly is more critical with respect to safety than classifying green lights correctly. Therefore, the performance objective for red lights will be higher than the one for green lights as far as safety is concerned.

The performance requirements also depend on the *scope*. For instance, in the railway domain, certification of systems relies on a concept called "GAME" (*Globalement Au Moins Equivalent* / Overall At Least Equivalent) that requires a new system to be at least as safe as the existing system it will replace. In our use case, this means that a misclassification may be considered acceptable if a train driver would have also misclassified the signal in similar conditions. Practically, this means that performance requirements in poor visibility environments may be lower than in normal operating conditions.

3) Examples of performance requirements

The table below provides examples of possible performance objectives for our use case:

	Example #1	**Example #2**	**Example #3**
Scope	All operating conditions All signals	All operating conditions Red signal	Poor visibility All signal
Metric	Accuracy	Accuracy	Accuracy
Obj.	99,9999%	99,99999%	99,999%
Conf. level	10^{-5}	10^{-6}	10^{-4}

B. Test stopping criteria

A test campaign must stop when the performance objective (e.g. 99,999%) is met with the target confidence level (e.g. 10^{-5}). Statistical test stopping criteria may be *distribution-independent* or *distribution-dependent*.

Distribution-independent criteria requires no assumption on the test dataset distribution with respect to the actual operational distribution, but are often intractable in practice, whereas the second one requires some assumptions.

The following notations are used in the next paragraphs:

- $\mathbb{P}(X)$ denotes the probability of an event X
- n denotes the size of the test dataset
- $\hat{p}_n$denotes the probability of an incorrect classification observed on the test dataset
- p denotes the actual unknown probability of an incorrect classification
- δ denotes the desired confidence level.

1) Distribution-independent criteria

Without assumption on the test dataset distribution, a generalization bound can be derived as explained in the course "Introduction to Statistical Learning Theory", in pages 191 and 192 in [18] : for some absolute constant C_1 and C_2 , the generalization bound reads [18]

$$\mathbb{P}\left(p \leq \hat{p}_n + C_1 \sqrt{\frac{VC}{n}} + C_2 \sqrt{\frac{\log(1/\delta)}{n}} \right) \geq 1 - \delta$$

In this formula, VC denotes the Vapnik and Chervonenkis dimension (VC dimension) [18].

Currently, these generalization bounds are often too loose to be usable. For example, if we take $\delta = 10^{-5}$ with $VC = 10^5$ (typical order of magnitude for a Deep Neural Network made of 10 layers and 10^4 weights [19]) the size n of the test dataset should be greater or equal to 10^{15}, which is impossible to achieve. Improving these generalization bounds is an ongoing research topic. Therefore, in this paper, this type of distribution-independent criteria is not retained.

2) Distribution-dependent criteria

If we assume that the samples of the test dataset:

(i) are independent and identically distributed (i.i.d)

(ii) have the same distribution as the operational distribution relevant to the scope of the requirement.

Note: Relevant operational distribution is defined with respect to the scope of the test. For example, if the scope is "all operating conditions, red traffic lights only", then the relevant operational distribution is the true operational distribution of red traffic lights, actually encountered by trains in operation.

Then the following relation holds, as given in lemma B.10 in page 427 in [20] :

$$\mathbb{P}\left(p \leq \hat{p}_n + \sqrt{\frac{2\hat{p}_n}{n} \ln\left(\frac{1}{\delta}\right)} + \frac{2}{n} \ln\left(\frac{1}{\delta}\right) \right) \geq 1 - \delta$$

This bound is much tighter than the previous one. For example, if we take $\hat{p}_n = 10^{-5}$ and $\delta = 10^{-5}$, the size of the test dataset should be $n \approx 10^6$, which is manageable.

But this bound requires a careful selection of the test dataset to make sure that the assumptions (i) and (ii) are met.

C. Test dataset collection and verification

In this section, we assume that the distribution-dependent stopping criterion is used. This choice implies a careful selection and verification of the test dataset in order to meet the

two assumptions: (i) the samples included in the test dataset are i.i.d and (ii) the test dataset distribution is identical to the relevant operational distribution [21].

1) Collection

In order to meet those two assumptions, we propose two approaches.

- **Random collection.** Data collected in operation are randomly sampled. Data collection and sampling are performed uniformly in order to reproduce the operational distribution. The challenge with this approach is to ensure that randomness and uniformity are effective, without bias.
  Example: for a test of "all operating conditions on a specific train line", all trains operating on the line are equipped with cameras during the whole year, and the test dataset is built by randomly sampling the collected data.
- **Planned collection.** The operational distribution is analyzed to identify all the relevant features and their frequency. Then, data collection is planned in order to gather a dataset with the same features at the same frequency as the operational distribution. The main challenge with this approach is to properly specify the operational distribution, without introducing bias in the specification.
  Example: for a test of "all operating conditions on a specific train line", the relevant features are identified: it includes weather conditions, light conditions, background type, traffic light types… Then the frequency of each situation is assessed, and collection is planned to have samples of each situation at the expected frequency.

2) Verification

The test dataset should be verified in order to check that it satisfies the assumptions. For the first assumption of independent and identically distributed (i.i.d), the verification approach differs depending on the collection approach:

- **Random collection.** The i.i.d assumption is satisfied by design, so the verification activity only aims at assuring that randomness was effective during collection.
- **Planned collection.** The i.i.d assumption may be more difficult to achieve with planned collection, and verification activities should assure that independence is effective.

In both cases, various tests exist to verify the i.i.d. assumption; such as autocorrelation plot, lag plot or turning point test [22]. None of these techniques can provide certainty, but they increase confidence.

The second assumption (consistency of the test dataset distribution with the operational distribution) is verified through an analysis of the consistency between the data collection conditions and the expected operational conditions. This analysis can be qualitative (expert judgement) and/or quantitative (comparison of features frequency).

Notes:

- *Expert judgment is used several times places in our testing process. However, this is not a way to bypass more formal and less subjective solution when available. In addition, expert selection and judgment still relies on a rigorous process, as shown for instance in* [23] *in the context of Assurance Cases.*
- *The operational distribution may evolve over time, for example when changes on the traffic signals or their environment happens. Therefore, the consistency of the test dataset distribution with the operational distribution may degrade over time. A mechanism to monitor the evolutions of the operational distribution should be implemented to identify such situation.*

D. Test results analysis

At the end of the test, two analyses are performed on the test results:

- **Performance analysis:** this first analysis is simple, as it only consists in verifying that the ML model meets its performance objectives. The distribution-dependent criteria defined in section B.2) is applied, and the result is *pass* or *fail*. If the result is *fail*, then the ML model should be retrained, and performance testing redone, with a new test dataset in order to avoid an iterative learning of the test dataset that would introduce bias in the test results. In case of reuse of the same test dataset, the distribution-dependent criteria is no longer valid, and other bounds should be applied, to account for the reuse.
- **Failure analysis:** additionally, if the ML model failed on some samples of the test dataset, and even if the overall performance is acceptable, each failure must be analyzed in order to find the root cause, and ensure that the underlying failure condition is local and not systemic. This failure analysis could be performed using explainability techniques [21].

V. ROBUSTNESS TESTING

A. Test objectives definition

According to IEEE Std 610.12-1990 , robustness is the degree to which a system or component can function correctly in the presence of *invalid inputs* or *stressful environmental conditions*.

We can see from this definition that robustness is twofold. In machine learning, among potential *stressful situations* we consider: adversarial attacks [23], worst cases or edge cases, and *invalid inputs* that are defined relatively to the distribution of the training data [24]. Edge cases will be discussed in the "Worst case section" because their study may be very specific to the application while adversarial robustness and out of distribution robustness share a lot of common point from one application to the other. Then we will focus on these two last situations in the following.

Adversarial attacks consist in transformations of the original data which are in general not visible for the human eye but which have the ability to change the response of the system. Synthesizing sophisticated adversarial examples and elaborating defense strategies are the topics of many research works [25] [26]. These invisible attacks, when used for robustness assessment, aim at providing a better understanding of the algorithm behavior in the neighbourhood of the test points. To do so, an optimization process is used to find a perturbation that provokes a decision change in the limit of a maximum perturbation radius is around a test data. If this process is often successful, the model is considered not robust.

Adversarial attacks can be visually visible and constructed to assess algorithm robustness [27] [28]. In the case of an outdoor application, like our use case, these visible perturbations can represent realistic situations like fog or damage caused to a signal.

Lastly, a technically very different approach but which goal is very similar to adversarial attack based approaches are

based on abstract interpretation of the ML component [29]. In this approach, a theoretical perturbation ball is defined around a testing point and is "forwarded" through a neural network using abstract interpretation theory. This ball is transformed by the network accordingly to its layers. At the end the transformed ball is compared with the decision boundaries. If it does not cross them, the network is robust to the tested amount of perturbation at this testing point.

The other concern about robustness corresponds to data which have not been learnt by the system, often referred as out-of-distribution (OOD) data, anomaly, outlier, or novelty [30] [31]. As a result, the behavior of the system on these data is often not predictable and needs to be evaluated. Nevertheless, it is often critical to assess that a data is in or out of the training distribution of the system. Consequently, it is necessary to develop tools to identify out-of-distribution data for observing the behavior of the system on these data only. The performance degradation should be progressive with respect to the distance between out of distribution data and in distribution ones. If it is the case the ML based system is considered as robust.

B. *Test stopping criteria*

As explained earlier robustness testing data are designed to explore potentially difficult situations. It is worthy to note that, benchmarks of model robustness are proposed in the literature [32]. However, in this document we consider evaluating robustness of a particular model dedicated to a particular use case. To do so, it is mandatory to be able to introduce a progression of the test cases difficulty in order to be able to measure the degree of robustness of the algorithm otherwise robustness analysis is nonsense. This graduation is easy to obtain in standard perturbation test cases:

- Noise gradation is based on noise level in dB,
- Signal frequency modification (e.g. blur) is controled by the applied filter transfer function profile,
- Geometric distortion are parametrized by geometric parameters like rotation angle.
- Percentage of occultation of a targeted object in the case of partial occultation perturbations.

It may be less obvious as for adversarial robustness testing. The natural choice is to monitor the amount of data modification through the attack radius ρ (the modified data must be comprised in a distance at most equal to ρ from an actual test set data). This idea has been extended to Wasserstein distance in [33]. Wasserstein attacks are interesting for robustness testing as they produce more realistic contents with respect to the data distribution than classical attacks; When studying the possibility of attacks in the physical world [34] and the ratio between efficiency and realization difficulty cannot be assessed by a radius criteria. However, we can draw some test stopping criteria

- Stop when: the algorithm resists to a predefined list of adversarial attacks with a radius less or equal to the natural noise amount (noise measures in the training data for instance).
- Stop when physical sticker attacks covering less than 20% (or what ever the specification) of the object of interest fail.

Concerning out of distribution (OOD) robustness, it is very difficult to establish. OODs are in the class of the "Known unknowns", that is "One knows OOD may occur, but does not know what it will be". Moreover, the notion of progressivity is very difficult to define and to obtain. However once again we consider that it is a very important characteristic that a robustness test must have. To fulfill this requirement we propose two approaches.

The first approach relies on an operational domain expert's analysis. The expert knowing well the operational domain, he should be able to express its limit and possible variations. From this analysis, out of definition data can be gathered or created through a simulation process.

The second approach computes the distance between the nominal data distribution and some gathered OOD data distributions. To do so one can estimate the Wasserstein distance between OOD sets and genuine distribution [35].

The test stopping criteria is based on the OODness measure (proposed by the expert or through the Wassertein distance). The test are stopped when the robustness limit of the system is encountered (observed through a performance drop) or the system performances are maintained for "far enough" OOD data. The rigorous gradation of all robustness tests allows to determine the ML component breaking points (similarly to mechanical testing). These breaking points can be compared to the system specification.

C. *Test dataset collection*

To evaluate robustness to data perturbation, a simple approach is to take all the available test data and to apply several adversarial transformations on it. A good practice is to list all the adversarial transformations that require to be tested (simple one like [23] or more complex one like [33]). A survey on adversarial attacks can be found in [26]. Then, it is necessary to tune their magnitude in order to make sure that the transformation remains realistic with respect to the application..

Building a dataset for OOD robustness assessment involves two steps. First, human experts may identify situations that should not be encountered by the system in the operational domain then that are not present in the training set. They can also imagine some shift in the operational environment of the system.

From these analysis the OOD datasets can be specified. OOD data are then gathered manually to represent these situations. Another method is to apply style transfer on the test data [36]. This allows evaluating the behavior of the system in realistic situations but with different appearance.

D. *Test results analysis*

It is impossible to prove that an ML algorithm is robust to every possible stressful situation.

To analyze the model robustness we observe the amount of perturbation that must be injected in the datasets before observing a significant performance drop. A finer observation of the results can enlighten in what kind of stress the system does perform most badly.

What can we do with the results? If the tests are performed at the final validation step, they should be compliant with specification to be considered as passed. If the tests are performed during algorithm development, adversarial attack can be introduced in the training data (adversarial training). If possible this approach can be done with OOD data but this will make more difficult OOD robustness assessment, new distribution of OOD data will need to be imagined and generated.

VI. WORST-CASE TESTING

A. *Test objectives definition*

Worst-case testing consists in observing the performance of the system in situations considered as being the "worst", or the most difficult, by experts of the operational domain. It is

important to note that this scale of difficulty is strongly related to human capabilities (processing, sensing, and actuation capabilities) and that they may not match with the actual "difficulty" for the ML system. For our use case, it means the worst cases as perceived by the train drivers, but also by the experts in Image acquisition and processing (used in the data acquisition chain).

Because performance testing only considers an average performance, and robustness testing activates the weaknesses of the ML model mostly independently from the operational domain, worst-case testing is a necessary complement to specifically observe the behavior of the model in difficult operational situations. Then, we are specifically interested here in the inputs that are likely to be generated in the real world but having the particularity of a low occurrence.

B. *Test stopping criteria*

Here the challenge is not to reach the exhaustiveness of the situations that could arise (i.e. the field is infinite), we focus on observing how the ML based system performs. It is clear that its average performances on these situations will be lower than what expected in the average conditions but our goal is to track to what extent of bad situation the system can go and if the operational limits we find are acceptable in practice.

Then stopping criterion is simple: Have all the worst cases identified by the experts been tested ?

C. *Test dataset collection*

Our use case is an outdoor perception one. Then first worst cases identified by an expert will concern weather, light conditions, scene clutter and etc. More specific situations like tunnel exit can also be identified as worst cases. Then the dataset must be defined from this comprehensive expert list of situations. These situations are frequently identifiable in the PHA (Preliminary Hazard Analysis). It is advised to also test in the neighborhood of the specified limits. Then, we propose a qualitative approach, which consists in classifying, based on an expert opinion, the most problematic situations to be managed for the train driver or the operator in general.

A non-exhaustive list could be:

- Test exceptional situations found in the ODD (with almost no sample in learning database)
- Based on expert knowledge, test critical situations (weather condition, occlusion, background, speed, vibration, line of sight, curve, etc.)
- Test safety critical scenarios (where missing the recognition leads to a critical situation)
- Test combination of already difficult cases, combination of robustness tests applied to worst-case images.

As explained earlier, these situations are rare by definition, then acquiring data representing them necessitate some dedicated effort. Real data will be acquired in the limit of feasibility and cost. To find a solution to this limitation, simulation based testing may be used [37] [38].

D. *Test results analysis*

The analysis of these tests allow determining the limit between the safe and unsafe domains. Beyond we could study the options of safety mitigation (e.g. failure mode, redundancy, etc.) and the way to monitor these critical situations, even redefine the operational domain. Then, the worst-case tests could provide a clue of confidence in the generalization in the real world and will highlight the limits of the safe domain of ML.

Some difficulties remains in:

- limited amount of real data representing worst cases,
- difficulty to guaranty the representativity of simulations.

VII. BIAS TESTING

A. *Test objectives definition*

Once it is ensured that the overall performance of the ML model is successfully tested with an unbiased test dataset, as described in section IV, it is also necessary to ensure that no bias is present in the decisions of the model itself. The model shall indeed use relevant features to output the decisions that have led to the recorded performances. In particular, the performance results shall not be obtained due to some attributes in the data (in images, color or texture, for instance) which possibly introduce a bias in the response of the system. This bias in the response corresponds to a flaw in the model's representation which is often due to a bias in the training set itself [39].

If the ML model returns biased responses, then it may be significantly more effective in some situations than others. This may not be acceptable: for example, the traffic signals detection should not be degraded on some tracks whereas it performs well on other tracks.

The absence of bias in the decisions of the model can be assessed using fairness and explicability techniques. Fairness techniques enable the detection of biased behavior of the model, whereas explainability techniques allow the analysis of the feature that most influenced the decisions made.

1) *Fairness*

Fairness is related to ensuring that the system applies a fair treatment to the data whatever the values of some of its attributes are [40]. The worst-case testing proposed in section VI, or the performance testing described in section IV used on subsamples of the dataset in order to compare the relative performances in different situations, may contribute to the detection of unfair data processing by the model.

2) *Explainability*

Due to bias in the training set, the ML model may also return its outputs based on irrelevant features of the input. A famous example is presented in [41], where a husky is classified as a wolf based on a snowy background. Using explainability techniques helps detecting such erroneous behaviors. Normally, such bias in a ML model should be detected through the performance testing approach described in section IV, with an unbiased test dataset. Additionally, explainability techniques could be used, such as LIME [41], or an occlusion sensitivity analysis as described in [42]. Nevertheless, caution should be taken with explainability techniques, since most of them have hyper parameters that greatly influence their results [43].

B. *Test stopping criteria*

Bias testing should be carried out on all relevant classes of bias, and stopped when the review of all these classes is completed. It is not expected that bias testing will enable the detection of unknown class of bias, because each testing techniques addresses a particular class of bias. Identification of all possible sources of bias cannot be achieved in practice. Therefore the identification of relevant classes of bias relies on expert judgement, in agreement with the certifying authorities.

Therefore, the following approach is suggested:

- **Identification of relevant classes of bias:** this activity should involve experts with various backgrounds, including data scientist, acquisition system expert, operator, etc.
- **Bias testing:** for each potential class of bias, one or several bias detection techniques should be applied to confirm or infirm the bias in the ML model. Fairness or explainability techniques, as presented in A, could be used depending on the type of bias. For each type of bias, the number of samples tested can be chosen based on the expected confidence interval, using criteria similar to the one described in section IV.B.2).
- **Stop:** bias testing stops when all known potential sources of bias have been tested on a sufficient number of samples.

C. Test dataset collection

The test dataset is defined by the potential sources of bias identified. For each potential source of bias, a representative test dataset should be collected, in a similar way as the one described in section IV.C.

For example, if the potential source of bias to be tested is "the ML model performs better on yellow lights than on red lights" (fairness problem), the dataset should extract two subsamples from the test dataset gathered in section 4.3: one subsample containing yellow lights, and one subsample containing red lights.

If the potential source of bias to be tested is "the ML model builds its decisions on the shape of the signal instead of its color" (explainability problem), then a dataset containing various combinations of signal shapes and colors can be used.

D. Test results analysis

For each confirmed source of bias, its acceptability should be checked with respect to the requirements. If the bias is confirmed and not acceptable, the ML model should be retrained to remove the identified bias(es). The analysis differs depending of the type of bias:

- **Fairness:** some degree of unfairness may be accepted. For example, it may be acceptable to have a ML model that performs better on red lights than on green lights, because it does not compromise safety. Therefore a careful safety assessment may support the acceptance of unfair ML model.
- **Explainability:** if the ML model makes decisions based on irrelevant features of the input, the ML model should not be accepted, except if the occurrence of such situation is proven to be rare enough.

VIII. RELATED WORKS

As of today, even if there is already a large number of ongoing initiatives about the *engineering* of Machine Learning in the context of critical systems, results are essentially focused on the identification of high-level challenges [1][2], or certification-level guidance [3] [45]. Our paper proposes solutions to address the challenge of verification and refines the relevant development phases down to practical engineering activities (definition of test stopping criteria, definition of dataset, etc.). However, our proposal remains partial for it only consider verification by testing and does not cover other verification means such as, for instance, formal verification.

A survey of 144 papers in Machine Learning Testing is given in [1]. This paper, which provides a comprehensive and structured analysis of the recent results concerning testing, identifies a large set of methods and tools to support this activity. Those methods and tools are some of the "building blocks" that we integrate in the testing strategy proposed in this paper. Our proposal is complementary to these papers in the sense that it proposes to organize in a sensible and practical way the various testing activities.

IX. CONCLUSION

To test safety-critical systems based on machine learning, we have presented an approach divided into four activities: performance assessment, robustness verification and worst-case testing. For each of these, we have given objectives and stopping criteria, and some solutions to collect the appropriate data.

In addition, we have illustrated our propositions in our use case of railway signal identification. As usual in machine learning domain, data are of particular importance. To ensure valid testing, we have underlined several considerations, which must be taken into account like the definition of the operational domain, or the detection of unintended biases in the datasets or in the model's decisions. Particularly for worst-case testing, due to the scarcity of worst-case situations.

As it can be seen, our approach intends to use complementary point of views of the system at hand to construct an efficient testing strategy.

REFERENCES

[1] IRT Saint-Exupéry; ANITI, "White Paper Machine Learning in Certified Systems," Toulouse, 2021.

[2] I. Stoica and et a., "A Berkeley View of Systems Challenges for AI," Berkeley, 2017.

[3] EASA, "Concepts of Design Assurance for Neural Networks (CoDANN)," 2020.

[4] 2. ISO/PAS, "Road Vehicles - Safety of the Intended Functionality (SOTIF)," Jan. 2019.

[5] F. Maleki, N. Muthukrishnan, K. Ovens, C. Reinhold and R. Forghani, "Machine Learning algorithm validation : from essentials to advanced applications and implications for regulatory certification and deployment," *Neuroimaging Clinics*, vol. 30, pp. 433-445, 2020.

[6] S. Picard, E. Jenn, C. Chapdelaine, B. Lefevre, C. Cappi and L. Gardes, "Ensuring Dataset Quality for Machine Learning Certification," in *10th IEEE International Workshop on Software Certification*, 2020.

[7] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barret and M.-J. Kochenderfer, "Algorithms for Verifying Deep Neaul Networks," *Found. Trends. Optim*, vol. 4, 2019.

[8] I.-2. RTCA, (DO-248C) Supporting information for DO-178C and DO-278A, Dec. 2011.

[9] I. s. 6. IEEE, IEEE Standard Glossary of Software Engineering Terminoloy (IEEE 610), 1990.

[10] T. Chen, F.-C. Kuo, H. Liu and P.-L. Poon, "Metamorphic Testing: A Review of Challenges and Opportunities," *ACM Computing Surveys*, 2018.

[11] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xy and T. Y. Chen, "Testing and Validating Machine Learning Classifiers byMetamorphic Testing," *Journal of system and sofware*, 2011.

[12] A. Torralba and A. A. Efros, "Unbiased look at dataset bia," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[13] A. Paullada, I. D. Raji, E. M. Bender, E. Denton and A. Hanna, "Data and its (dis) contents: A survey of dataset development and use in machine learning research," in *Advances in Neural Information Processing Systems Workshop : ML Retrospectives, Surveys & Meta-analyses (ML-RSA)*, 2020.

[14] Y. Li and N. Vasconcelos, "Repair : Removing representation bias by dataset resampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[15] N. Jaipuria, X. Zhang, R. Bhasin, M. Arafa, P. Chakravarty, S. Shrivastava, S. Manglani and V. N. Murali, "Deflating dataset bias using synthetic data augmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

[16] I. J. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press.

[17] M. Moukari, S. Picard, L. Simon and F. Jurie, "Deep Multi-scale architectur for monocular depth estimation," in *ICIP*, 2019.

[18] O. Bousquet, U. von Luxburg and G. Rätsch, Advanced Lectures on Machine Learning : ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures, Springer, 2003.

[19] P. L. Barlett, N. Harvey, Liam, C. Liaw and A. Mehrabian, "Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks," *Journal of Machine Learning Research*, 2017.

[20] S. Shalev-Shwartz and S. Ben-David, Understanding machine learning : From theory to algorithms, Cambridge University Press, 2014.

[21] D. Bau, B. Zhou, A. Khosla, A. Oliva and A. Torralba, "Network Dissection:Quantifying Interpretability of Deep Visual Representations," MIT, 2020. [Online]. Available: http://netdissect.csail.mit.edu/final-network-dissection.pdf.

[22] J.-Y. Le Boudec, Performance evaluation of computer and communication systems., EPFL Press, 2010.

[23] P. J. McGee and J. C. Knight, "Expert judgment in Assurance Cases," in *10th IET System Safety and Cyber-Security Conference 2015*, Bristol, UK, 2015.

[24] I. J. Goodfellow, J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[25] C. M. Bishop, "Novelty detection and neural network validation," in *IEE Proceedings-Vision, Image and Signal processing*, 1994.

[26] A. Athalye, L. Engstrom, A. Ilyas and K. Kwok, "Synthesizing Robust Adversarial Examples," in *International Conference on Machine Learning (ICML)*, 2018.

[27] X. Yuan, P. He, Q. Zhu and X. Li, "Adversarial Examples : Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805-2824, 2019.

[28] K. Pei, Y. Cao, J. Yang and S. Jana, "DeepXplore: automated whitebox testing of deep learning systems," *Commun. ACM*, vol. 62, no. 11, p. 137–145, Oct. 2019.

[29] A. Odena, C. Olsson, D. G. Andersen and I. Goodfellow, "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing," 2018.

[30] T. Gehr, M. Mirman, D. Drachsler-Cohen and P. Tsankov, "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," in *2018 IEEE Symposium on Security and Privacy*, 2018.

[31] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection : A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[32] H. Wang, M. J. Bah and M. Hammad, "Progress in Outlier Detection Techniques : A Survey," *IEEE Access*, vol. 7, pp. 107964--108000, 2019.

[33] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *ICLR*, 2019.

[34] K. Wu, A. H. Wang and Y. Yu, "Stronger and Faster Wasserstein Adversarial Attacks," in *ICML*, 2020.

[35] K. Eykholt, I. Evtimov,, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmat and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *CVPR*, 2018.

[36] C. Villani, Optimal Transport: Old and New, Springer, 2009.

[37] L. Gatys, A. S. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks.," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (pp. 2414-2423).*, 2016.

[38] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh and M. Vazquez-Chanlatte, "VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems," in *31st International Conference on Computer Aided Verification (CAV)*, 2019.

[39] M. Mousavi, A. Khanal and R. Estrada, "AI Playground: Unreal Engine-based Data Ablation Tool for Deep Learning," 2020. [Online]. Available: https://arxiv.org/pdf/2007.06153v1.pdf.

[40] Q. Zhang, W. Wang and S.-C. Zhu, "Examining cnn representations with respect to dataset bias," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[41] E. del Barrio, F. Gamboa, P. Gordaliza and J.-M. Loubes, "Obtaining Fairness using Optimal Transport Theory," in *International Conference on Machine Learning (ICML)*, 2019.

[42] M. T. Ribeiro, S. Singh and C. Guestrin, "" Why should I trust you ?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.

[43] C.-H. Cheng, C.-H. Huang, H. Ruess and H. Yasuoka, "Towards dependability metrics for neural networks," in *The 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2018.

[44] N. Bansal, C. Agarwal and A. Nguyen, "SAM : the sensitivity of attribution methods to hyperparameters," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[45] DGA, "Guide méthodologique pour la spécification et la qualification des systèmes intégrant des mo-dules d'intelligence artificielle," 2020.