ISSN 2754-1118 (Online)
ISSN 2753-6599 (Print)

Towards Defect-based Testing for Safety-critical ML Components

Amit Sahu and Carmen Carlan

Edge Case Research GmbH, Munich, Germany

Abstract

The input space of Machine Learning (ML) components used in safety-critical applications is complex. Testing such components on exponentially large datasets that cover all potential real-world situations to meaningfully measure performance metrics, such as the false negative rate, is infeasible due to practical restrictions. Consequently, we must limit the test data while adequately covering critical input data points. Inspired by defect-based software testing, a method for specifying adequate test cases for software components, we propose a process for collecting adequate test data for ML components. Concretely, we systematically employ different existing ML data quality metrics, and methods for enhancing the test data, to uncover critical scenarios where the ML component may be less performant. We exemplify the usage of our process in two case studies, each involving an ML component implementing different functionalities, i.e. stop sign recognition, and railway track segmentation.

1 Introduction

Automated Driving (AD) systems have an unpredictable and complex operational domain. Therefore, AD functions are (partially) implemented by Machine Learning (ML) components, which use training data to approximate the target function rather than using the specific requirements.

ML components that are used to implement safety-critical tasks must demonstrate that they are sufficiently performant. To measure the performance of ML components, different metrics like accuracy or false negative rate are measured on test datasets. However, it is difficult to extrapolate the performance results obtained while exercising ML components with test datasets, to the entire operational domain. For example, a test dataset may be biased (Pagano et al. 2023), imbalanced (Ashmore et al. 2021), or may not be representative of the operational input domain (Zendel et al. 2015). Also, critical scenarios may not be well represented (Cheng et al. 2018b). Here, critical scenarios are scenarios in which the poor performance of an ML system leads to invalidation of system safety goals. Further, test datasets may not have good coverage of the data distribution in the feature subspace (Mani et al. 2019). Consequently, there is little confidence in the performance measured while exercising the ML component with test datasets when used as evidence in safety arguments.

While current standards and guidelines, such as ISO 21448:2022, require that the training, testing, or validation datasets used are good approximations of the real world, they do not provide practical guidance for how to collect such datasets.

In traditional software testing, to measure the performance of components, good test cases are defined and created. For example, in the context of defect-based testing, a "good" test case is defined as a test case that, when executed, has the potential of revealing system defects (Pretschner 2015). Defect-based software testing builds defect models to capture potential defects. These are defined and used to search for good test cases. In the ML literature, with similar objectives, metrics for measuring the quality of ML data, such as scenario coverage or domain shift, have been proposed. However, there is still no indication of how using these ML data quality metrics supports the detection or reduction of ML component defects.

In this paper, inspired by concepts from defect-based software testing, we propose a novel method for collecting, generating, or selecting ML test data. New ML test data may be collected from sensors and new data points may be selected from newly collected ML test data. Further, specific syntactic test data points may be generated, for example, by setting parameters in a simulator. Further, we provide requirements on new good test data points. These can be used either to filter (data selection) or search for (data collection) test data points. We first define what an adequate test dataset is and then specify a systematic process to collect adequate test datasets. To ensure that different types of ML defects can be identified during testing, our method recommends that different ML performance and data quality metrics are meaningfully combined for collecting the test dataset. To this end, we propose investigating which ML defects may be identified, eliminated, or mitigated using a specific ML metric. We showcase the execution of the process given an exemplary set of ML data quality metrics available in the Neural Network Dependability Kit (NNDK) , an open-source toolbox supporting safety engineering of neural networks.

Next, in Section 2, we discuss concepts upon which our work builds, such as software defect-based testing, ML-specific defects, and ML metrics. Then, in Section 3, we elaborate on our process for collecting adequate test data. In Section 4, we exemplify how a set of selected ML data and performance metrics can be complementary used to collect test data. We continue in Section 5 by exemplifying the usage of our method on two case studies, whereas in Section 6, we discuss the open challenges for using ML metrics as evidence in safety arguments. Section 7 positions the contributions of our work in the current state of the art. Finally, in Section 8, we summarize the contributions of this work and discuss the next steps.

2 Background

2.1 Defect-based Testing

Our approach for collecting adequate test data for ML safety-critical components is inspired by defect-based testing from the software domain. "*A good test case reveals a potential defect with good cost-effectiveness*" (Pretschner 2015). Defect-based testing argues this concept in a systematic manner. Defect-based testing should be applied when random testing does not support the identification of sufficient system defects, being most efficient when a set of recurring defects can be defined. A software defect is an error, a fault, or a failure within the source code or within the operating environment of the software, which causes a deviation from the desired behaviour of the system.

In defect-based testing, defect classes need first to be defined. A defect class specifies a deviation from the system specification in a set of behaviour descriptions. Examples of software defect classes are division-by-zero, null pointer dereferencing, stuck-at-one, and

array index overflow. Whereas these are examples of domain-independent defect classes, domain-specific defect classes may also be specified. For example, out of distribution (but within the Operational Design Domain (ODD)) is a defect specific to data used for training ML components. The ODD, as defined by SAE J3016_202104, describes the "*operating conditions under which a given driving automation system, or feature thereof, is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics*".

Second, based on the defined defect classes, defect models for these classes shall be specified. Defect models are single instances of defect classes. They may be used to search for test cases, as they partition the input domain of the system into inputs that lead to incorrect behaviour, i.e. the defect domain, and inputs that lead to correct behaviour.

Third, to determine the "goodness"/adequacy of a test case, fitness functions may be used (Kolb et al. 2021). A fitness function assigns to a test case a fitness value, indicating which test cases challenge the system, i.e. can detect system defects. The fitness function must be specific to the use case. In the software domain, the fitness function could be running the code in multiple scenarios and counting the number of errors generated by a specific input. For example, inputting "null" values into a calculator program and doing addition, subtraction, division, and multiplication scenarios. The more errors are generated, the better the test case, and thus, the fitness function should assign a high quality to the respective input. In ML domain, a fitness function may measure accuracy, false negative rate (FNR), mean Intersection over Union (mIoU), precision, or recall.

2.2 Defects Specific to ML Components and ML Metrics

Padilla et al. (2020) identify a set of defects related to the performance of ML components. First, a False Positive (FP) specifies the incorrect detection of a non-existent object or the misplaced detection of an existing object. Second, a False Negative (FN) indicates an undetected ground-truth bounding box or class. Other ML performance defect classes can be derived from ML performance metrics. Examples of performance metrics include precision, recall, per class accuracy, per class confidence score, and robustness against Out of Distribution (OOD) samples.

ML data defect classes can be derived from ML data metrics. An ML data defect class is the fact that the considered ML data does not meet the target value defined for a specific ML data metric. There are several ML metrics proposed by the ML Engineering community (Ashmore et al. 2021) (Tambon et al. 2022) (Willers et al. 2020) scoping the detection and/or the reduction of FPs and FNs. Examples of data quality metrics are k-projection coverage, neuron activation pattern, perturbation loss, scenario coverage, data augmentation (Cheng et al. 2018b), unknown behaviour in rare critical situations, adversarial attacks (Goodfellow et al. 2015), layer-wise relevance propagation (Montavon et al. 2019) for transparency and explainability in neural networks, and sanity checks during model development.

3 Proposed Process for Collecting Adequate Test Data

We use the concepts from defect-based software testing to develop a systematic technique for collecting, generating, or selecting adequate test data for ML-based components that

implement safety-critical functionality. The technique synthesizes test data samples from ML defect classes.

As there is no generally accepted definition for adequate test data for ML-based components, in the same line of thought as defect-based testing, like good test cases, we define a "good test dataset" as a dataset that will exercise the system with problematic inputs, scoping at uncovering defects of ML components. As a result, the ML performance values measured during testing will most probably initially suffer a decrease. Whereas a good test dataset uncovers possible defects in the system, an adequate test dataset will uncover defects belonging to all identified safety-critical defect classes. Safety engineers may argue about which defect classes are safety-critical using safety arguments. Such safety arguments shall show how component-level performance measurements indicate the satisfaction of system-level safety goals.

Next, we specify a four-step process for collecting test data by combining different available ML metrics and methods. The scope of this is to maximize the number of ML defect classes addressed by the test data. ML metrics and methods are used to measure the quality of data, or to add new test data points to reach the targeted values (as per requirements) for the metrics.

Step 1: Select an ML data metric/method and identify the addressed ML defect class(es);

Step 2: Measure the metric/execute the method. If the target value is not met, collect new data points;

Step 3: To measure the quality of the newly added data points, apply one or more fitness functions. A fitness function assigns to the test dataset a value obtained from the measurement of an ML performance metric, such as a false negative rate. If the newly collected data points are effective test data points, then the measured performance value will initially drop, meaning that system defect models are detected during the testing of the system given the collected test data points. This step will allow the engineers to control the size and quality of the test dataset by acting as a filtering mechanism with a minimum threshold on the fitness function;

Step 4: Optionally, to increase the probability of uncovering defects belonging to different defect classes, select a new data metric/ method. Then, execute again Step 1 to Step 3 for the newly selected metric/method, and, if needed, also Step 4.

Even after the addition of data points collected while using the ML metrics, there will still be a considerable number of data points that will not be tested, and for which there is no guarantee for how the system will behave. To compensate for the data points in the input space that are not covered by the test dataset, a filtering mechanism to restrict ML output in such scenarios is needed. A filtering mechanism would ensure that defect models belonging to unknown, or known but not addressed, defect classes are identified during system deployment. For example, one such filtering mechanism could be implemented by runtime monitors, which enable the detection and handling of defects at runtime.

4 Application

A pre-requirement of our technique is to have a pre-selected set of ML data quality and ML performance metrics. While the adequacy of the data depends on the selected metrics, the selection of the metrics is out of the scope of this work, and needs to be further explored in the future. In practice, system developers use ML metrics for which they have tool support or about which they have the most expertise. Further, safety engineers should

reason about how the selected metrics provide sufficient and trustworthy evidence about the data quality in the system safety case. In Section 7, we discuss state-of-the-art ML metrics that, with the help of our technique, could be combined to collect good test data.

Other pre-requirements are the existence of an initial test dataset and system-level safety requirements. The system-level safety requirements are needed to derive the target values for the selected ML metrics.

Next, we explain how we combined a concrete set of selected ML metrics to evaluate and enhance test datasets. In this example, we consider the ML metrics proposed in the Neural Network Dependability Kit (NNDK) (Cheng et al. 2018a), an open-source toolbox to support data-driven engineering of neural networks for safety-critical domains. First, to check whether the initial test dataset sufficiently covers the ODD, NNDK provides tool support for measuring the k-projection scenario coverage during the data management development stage. To this end, an ODD of the system shall be specified. Second, during the ML model learning and validation development stages, NNDK can be used to measure the perturbation loss metric, which assesses the vulnerability of the ML component to noise. Third, the neuron activation pattern coverage can be used during the model verification (testing) stage for detecting missing feature combinations. Fourth, NNDK makes available runtime monitors that can be used during model deployment (system operation) to detect new system defects based on data from the operating context.

In Figure 1, we illustrate how to argue about test data adequacy based on evidence generated while executing our proposed four-step method given the NNDK metrics. As proposed by Hawkins et al. (2021), such an argument shall be part of any argument about ML safety assurance. The argument is based on a combination of design-time and runtime evidence. Each tree branch uses a metric used in one of the development stages. In this work, the considered development stages are data management, ML model learning and validation, model verification (testing), and model deployment. In the figure, we depict the data points within the test dataset as red circles and the data points in the rest of the input space as green triangles. The figure shows that green triangle points shift to red circle points as new test data points are created while measuring and optimizing different ML metrics. For example, applying adversarial attacks results in new red circle points.

First, in Step 1 of the technique proposed in Section 3, we identify the ML defect class addressed by the NNDK k-projection scenario coverage metric. Given a test dataset and the ODD of a system, the k-projection scenario coverage metric identifies if the dataset fails to cover the ODD. The metric uses the meta-labels (other than the prediction class) of the data points to check the coverage of different combinations of scenarios and situations.

Then, in Step 2, the metric is to be measured for a given ML model and a test dataset, and the measured value is to be compared against the target value defined based on the system requirements. If the measured value does not meet the target value, the test dataset must be enhanced. Namely, to reach the target value for k-projection coverage, new data points with new combinations are to be added to the test dataset. The metric can suggest ideal data points with the most potential in increasing the metric for example, sunny with snow and a rainbow. However, these suggestions need to be filtered by the probability of real-life occurrences and acquiring difficulty. In Figure 1, we show how new data points are added to meet the target value.

In Step 3, we measure the quality of the new test data points using a fitness function (performance metric) to further filter the high-quality test cases as good test cases cause defects. If the measured values of the selected fitness function are less when the system is exercised with the new test data than when the system was exercised with the initial test

data, then critical scenarios in which the system does not perform well (high-quality) are present in the new data points.

In Step 4, we iteratively consider other ML data metrics that can be measured with NNDK, and for each of these metrics, we execute again Steps 1 to 3. The analysis of the newly uncovered critical scenarios may also be used to improve the system's performance in the next development iteration.

To check another ML defect class, namely the vulnerability of the ML component to noise, similarly to fault injection software testing methods, in the domain of ML verification, it is commonly accepted to use data augmentation and the addition of noise (Adversarial (Goodfellow et al. 2015), Bayesian, etc.). Since the datasets are only supplemented and not reduced, the domain coverage of the dataset is not affected.

Inspired by software testing, where structural and requirements coverages are computed, neuron activation patterns can be used for ML components to detect (neuron) features not contained in the dataset. More complex approaches of coverage testing like Mani et al. (2019) could also be used. The NNDK neuron coverage, like k-projection coverage, can suggest characteristics of new samples that will increase the test coverage.

Since these steps can generate different numbers of new test cases, a balancing based on the total number of all defect classes should be considered. As the final step, the NNDK runtime monitor can be added to the system architecture with the scope of detecting OOD data points. The ML component should be restricted from computing these points, as the output is unknown. Data points (during operation) still passing the runtime monitor, but failing the system (high FNR), could be considered as a measure of the leftover risk of using the ML component inside the system. This is the final filter to build an adequate test dataset (of good test cases) for the next iteration.

5 Evaluation

5.1 Preamble

Next, we demonstrate that the usage of NNDK metrics for enhancing test datasets, as discussed in Section 3, indeed adds value to the test datasets, meaning that the enhanced datasets uncover more defects. To this end, we apply the process proposed in Section 4 in two small systems. The first is a stop sign recognition component inside an autonomous vehicle system, and the second is a railway track segmentation component for an autonomous train.

The Stop Sign Recognition component is tasked with recognizing stop signs on the road for an autonomous vehicle. The basic ODD is all weather conditions, lighting, and size. In a real environment, the ODD will be more complex and include combinations of scenarios, for example, sign age, and condition. We are limited by the dataset only to images of stop signs in different scenarios.

The Track Segmentation component is responsible for identifying the pixels of the railway track in images with cityscapes. Hence, in addition to weather, lighting, and size the ODD will require different combinations of the scenarios happening in the environment. We considered labels like 'road', 'sidewalk', 'construction', 'tram_track', 'fence', 'pole', 'traffic_light', 'traffic_sign', 'vegetation', 'terrain', 'sky', 'human', 'rail_track', 'car', 'truck', 'trackbed', 'on_rails', 'rail_raised', 'rail_embedded', 'background'.

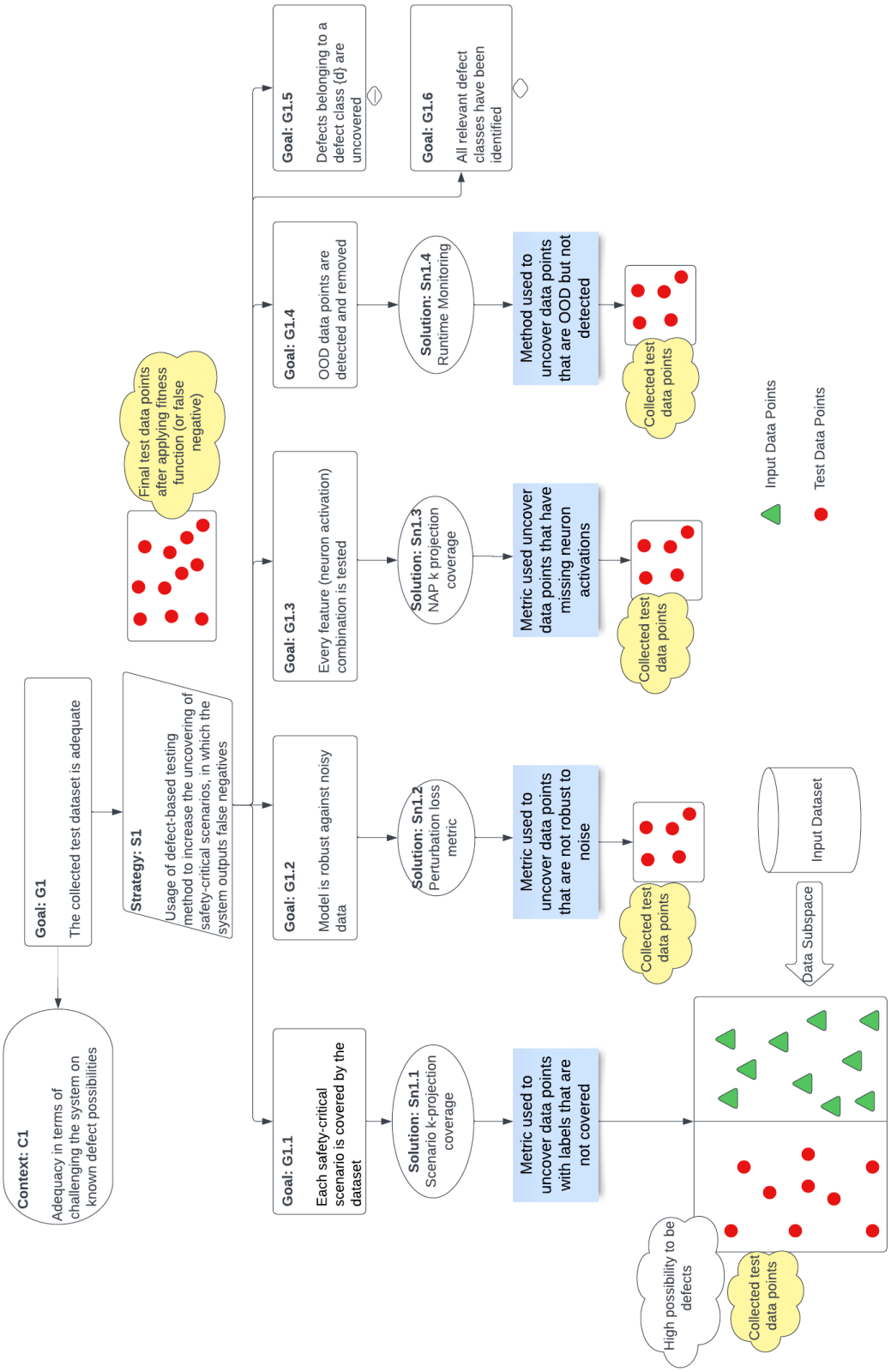

Figure 1~ Argument Explaining the Collection of Test Data Using NNDK Metrics

One pre-requisite of our process is the existence of an initial test dataset. Consequently, for each of the two case studies we consider in this work we first selected a testing dataset, which we used to measure selected ML performance metrics. Then, we applied our

process for collecting good test data points, and we measured again the selected ML performance metrics. The values of the ML performance metrics on the new enhanced test dataset suffered a decrease (Table 1 and Table 2). This means that the enhanced test dataset contains critical data points, that uncover the component's functional insufficiencies.

5.2 Initial Test Datasets

Whereas for the Stop Sign Recognition component we used as initial test dataset the German Traffic Sign Recognition Benchmark (GTSRB) (Stallkamp 2021), for the railway track segmentation component we used the RailSem19 dataset (Zendel et al. 2019).

The GTSRB dataset consists of 43 classes of traffic signs, and the images have varying light conditions and rich backgrounds.

The RailSem19 dataset for the Track Segmentation model consists of 8500 unique images from the ego-perspective of rail vehicles (trains and trams). The dataset provides extensive semantic annotations: geometry-based (rail-relevant polygons, all rails as polylines), and dense label maps with many road labels in Cityscapes.

5.3 Considered ML Performance Metrics

To measure the performance of the stop sign recognition component, we used the FNR performance metric. In the case of stop sign recognition, false negatives may cause more severe accidents than false positives.

For railway track segmentation, we used mIoU to measure the performance of the ML component. Since it is a segmentation model that assigns each pixel a classification, intersection and union with the ground truth is the standard performance metric.

5.4 Experiments

Table 1 and Table 2 show the values measured with NNDK for each of the considered case studies. Note: In Table 1, FGSM is the Fast Gradient Sign Method.

Table 1 ~ NNDK Results Measuring the Quality of the GTSRB Dataset

Data QualityMetric	**Basic Iteration (Performance metric)**	**The 5-step Process (Performancemetric)**
Perturbation Loss Metric (Snow)	3.33% (FNR)	32.6% (FNR with snow)
Adversarial Attack (FGSM)	N/A	25.82% (FPR)
Neuron k-projection coverage	3.33% (FNR)	98.33% (FNR with complete synthetic coverage)
Runtime Monitor	N/A	7.9% (FNR with Snow) and 22.5% (FPR with FGSM)

In Table 2, GN is Gaussian Noise.

Table 2 ~ NNDK Results Measuring the Quality of the RailSem19 Dataset

Data QualityMetric	**Basic Iteration (Performance metric)**	**The 5-step Process (Performance metric)**
Scenario k-projection coverage	57.68% (mIoU)	43.87% (mIoU with Fog)
Perturbation loss (GN)	57.89% (mIoU)	4.83% (mIoU with GN)

Initially, the FNR on the GTSRB dataset was 3.33%. We then applied snow perturbation, which was a missing label in the covered scenarios, to increase k-projection coverage. The FNR of the validation dataset dropped to 32.6%, indicating that snow is a limiting factor for the performance of the ML component. Further, we enhanced the test dataset with data points including iterative Fast Gradient Sign Method (FGSM) adversarial attack. While exercising the ML component with the new data points, the false positive rate was 25.82%. In other words, images in the dataset were attacked with noise to make them falsely classify as a stop sign and succeeded in 25.82% of the cases.

Modifying the dataset images synthetically to increase the Neuron Activation Pattern (NAP) metric resulted in an FNR of 98.33% on the synthetically created dataset. The synthetic images had neuron activation patterns that were missing in the test dataset. Hence, augmenting the test dataset with the synthetic images resulted in complete 2-projection coverage. The complete synthetic image creation process is explained in the NNDK paper (Cheng et al. 2018a).

Finally, we applied a runtime monitor based on the NAP monitoring technique from NNDK. After applying the runtime monitor, the FNR of the leftover images was reduced to 7.9% for snow perturbation and 22.4% for iterative FGSM. These results indicate the potential of the NNDK ML metrics to uncover critical scenarios. Given only these metrics, collecting the test data points using all the metrics resulted in an improved dataset, which helped with finding issues with the ML model. However, the adequacy of the test dataset is dependent on the data quality metrics as evidence. In the future, we plan on building an argument about the adequacy of a test data using the NNDK metrics.

For our second experiment, we trained a modified DeepLabV3Plus model (Chen et al. 2018) from github2 on the RailSem19 dataset. We modified the model parameters to keep the model within a limited GPU memory of 8 GB. The model was trained on 4200 images, and resnet34 encoder with ImageNet weights. This resulted in a mIoU performance score of 57.68% on 1000 test images.

On the RailSem19 dataset, the k-projection coverage was computed using the labels mentioned in the ODD. As an initial assessment k=2 was taken as an input for the metric. The 2-projection coverage using ODD labels gives us a high value of 99.71% (712 out of 714 two-label combinations). However, these labels were manually selected. Instead, to get a value confidently reflecting the current performance of the ML component, all relevant labels for the operational domain should be added. For example, the weather labels in test images included sunny, rainy, and cloudy weather but missed fog. Consequently, the synthetic noise of fog was added to all the test images, and then the mIoU was computed on the same dataset. The score dropped from 57.68% (without fog) to 43.87% (with fog).

To argue about the performance of the considered railway track segmentation model under sensor noise, the performance under Gaussian noise (GN) with *std = 0.1* was also

computed. The performance dropped from 57.89% (without GN) to 4.93% (with GN). As in the case of the first use case, these results show the potential of the NNDK ML metrics in uncovering critical scenarios that need to be tested to make the system more performant.

As data collection is a tedious process, for exemplification, in the two considered case studies, the data collection process was addressed as a data augmentation process. This has been done due to the reduced resources we had while conducting the case studies. Still, when implemented in a real-life project, our process assumes the collection of data by testing on a test track or operating in a real-world environment. Test data may also be collected from simulations or standard benchmark datasets.

6 Discussion

Continuous identification of critical test data points. Finding all defects during system development is difficult, given the uncertainties due to the complexity of the input space, the lack of interpretability of the ML components, and the complexity of the operational environment. Each step in the lifecycle of an ML component aims at identifying and mitigating ML defects to reach the required performance. Consequently, test data collection is continuous throughout the system lifecycle, meaning that the different considered ML metrics may be applied at different development stages of an ML component. Further, at runtime, given operating scenarios that were not considered during design time, system defects may be uncovered. In this work, we assume a continuous system development process, during which data samples are continuously added to the test dataset, covering newly identified operating scenarios.

Focus on test data. Defect models provide a systematic process to uncover new good test cases. Hence, our process is to augment the original test dataset. We consider adequacy based on knowledge provided by the metrics. Other aspects of the test dataset are not considered. The method can also be applied for validation or training datasets, but the intention of the process is to test the system with more difficult scenarios. Hence, the learning and validation benefits must be subjectively analysed as per the use case.

The importance of using adequate ML metrics. ML metrics are used during the development of the ML model to measure different properties of ML components, such as robustness against noise. However, these metrics do not guarantee that the model satisfies such properties. Since our process is dependent on the used ML metrics, it is as good as the used metrics. For example, in the second case study, replacing neuron k-projection metric with coverage testing (Mani et al. 2019) would have resulted in a higher-quality test dataset. Trusting the metric and the threshold defined as the target value acquired by experimentation on a variety of datasets is currently the only choice. While the usage of formal methods (Beyene and Sahu 2020) (Katz et al. 2017) has been researched to reach some guarantees, the application of such methods is only feasible when the ML models are small (a few thousand neurons), whereas ML models implementing safety-critical functions are usually considerably big, with millions of neurons and parameters (Redmon and Farhadi 2018). Safety arguments could be used to reason about the adequacy of the measured ML metrics to be used as safety evidence.

Additional general metrics. While specific ML methods may not be related to data directly, some methods like data augmentation, or adversarial attacks, are useful for identifying the defect models, as we show in our use cases. Adversarial defects point us to differences in features used by ML models as compared to humans. Since they are

artificially created, the possibility of their occurrence in the ODD should be validated and then used accordingly (Dilmegani 2024).

Data synthesis instead of data collection. While our process recommends the collection of new test data, this data collection process is usually tedious. Consequently, while conducting our case studies, we replaced data collection with data synthesis (namely, data augmentation). Further research is to be done on whether synthesized test data could also be used.

7 Related Work

7.1 Defining Test Data Adequacy

Having a good quality test dataset is an important prerequisite for measuring the performance of an ML component. Kim et al. (2023) define the adequacy of test datasets in terms of the degree of "out-of-distribution-ness" of a given input. The more the model gets "surprised" from the test input (compared to training input), the more adequate the test dataset. In contrast, Deepgauge (Ma et al. 2018a) uses minimum and maximum neuron activation values and k-multisection neuron coverage for gauging the adequacy of test datasets. Sun et al. (2018a) use Modified Condition/Decision Coverage (MC/DC) (Kelly et al. 2001) to define four novel criteria to structure features and semantics of deep neural networks, which are used to measure the adequacy of the test dataset. The main idea behind this method is that all the conditions that contribute to a decision must be tested.

These state-of-the-art approaches do not contradict each other, but are complementary. Whereas these approaches address single characteristics of the test dataset, we propose a higher-level definition of test data adequacy, stating that an adequate test dataset can detect all possible types of system defects. To enable the creation of adequate test datasets, we propose a four-step data collection process employing different ML methods and metrics. While the approach proposed by Kim et al. (2023) can be seen as an ML data metric to be used in our process to evaluate and enhance the collected data, the approach proposed by Sun et al. (2018a) and Ma et al. (2018a) propose ML performance metrics that can be used as fitness functions.

7.2 Systematically Searching Through the Input Space

In the literature, different approaches for generating test cases have been proposed. Such approaches may be used to collect ML test data.

CV-HAZOP (Zendel et al. 2015) combines guide words and parameters to systematically investigate the critical scenarios at every location in the system. The parameters refer to the physical and operational aspects of the sub-component configuration, whereas a guide word is a short expression to express the deviation from the intent of the design or process. For example, light source intensity (parameter) and "More" (guide word) will result in "Overexposure of lit objects" as a suggested hazard.

Zhao et al. (2021) introduced a Reliability Assessment Model (RAM) for ML classifiers. First, system-level safety targets are broken down into component-level requirements and claims supported by reliability metrics (Claims Arguments Evidence structure). Second, the input domain space is split into cells like a geometrical grid. Third, the operational profile, i.e. the possible output map of the component, and the robustness of these input

cells are measured, and the measurements are then combined to estimate the reliability of the claim.

Mani et al. (2019) proposed a method for test case generation using the quality aspects of the feature space: equivalence partitioning, centroid positioning, boundary conditioning, and pair-wise boundary conditioning are also provided.

Approaches modified from traditional software testing to accommodate them for ML models like mutation testing (Ma et al. 2018b), concolic testing (Sun et al. 2018b), illumination search (Zohdinasab et al. 2021) are different defect classes.

We propose a systematic process for using well-known methods to uncover critical scenarios and test cases. There are a number of these methods and new ones are constantly published.

Dola et al. (2024) applied combinatorial interaction testing to the latent space of generative models and can generate rare and fault-revealing test inputs. If applicable, this would be a good replacement for the NNDK neuron activation pattern. Hirschle et al. (2023) proposed a workflow to generate high-level scenario descriptions that are parameterized with the operational envelope data of the ML system. Synthetic data can then be simulated based on these parameters. This can be a replacement for NNDK k-projection coverage. Hartjen (2023) proposes a method for scenario management and their semantic classification.

In Alshareef et al. (2023), feature importance weights, that are weighed as per the contribution on the model output, are used to measure the coverage of the test set. This also facilitates the generation of additional test cases. This approach is a good replacement for the NNDK neuron activation pattern.

Whereas the above approaches propose test cases using weights, latent space, or scenario classification methods, our paper proposes an approach for combining (many) well-known methods to build an adequate test dataset. Individually all these approaches supplement our approach as individual metrics.

8 Summary and Future Work

Our work was motivated by the problem of collecting adequate test datasets for ML components implementing safety-critical functionality. To solve this problem, we defined a systematic, four-step data collection process, which uses state-of-the-art ML data and performance metrics and concepts behind defect-based testing to operationalize the acquisition of high-quality test data points. The proposed approach aims to be complementary to other test approaches, such as unguided and guided random testing. We showcased the usage of the process in two small case studies implementing different functionalities, i.e. stop sign recognition and railway track segmentation.

Next, we plan to apply the proposed technique in a more complex case study, employing a higher number of ML metrics.

Another line of future work is to provide technical solutions for executing each of the steps in the proposed process. To execute Step 2, novel techniques for generating new inputs and their corresponding labels are needed. To support the execution of Step 3 and 4, we plan on proposing a selection mechanism as an aid for choosing the ML metrics and methods used in the process. Here, one way to combine more systematically different ML metrics to collect test data points is to cover the known limitations of one ML metric with

other metrics. For example, one limitation of the k-projection coverage metric is that it does not ensure that the dataset is balanced. Namely, the metric does not provide information about a specific scenario's number of data points, i.e. even one data point can satisfy k-projection coverage for a critical scenario. This could be easily remedied by checking the proportion or representation of each scenario in the number of input samples. After that, data balancing techniques like oversampling, under-sampling, or class weight can be used.

Further, designing run-time monitors that can detect scenarios not covered by the dataset is a challenging task, which can be further investigated.

Acknowledgments

This research received funding from the Federal Ministry for Economic Affairs and Climate Action (BMWK) and the European Union under grant agreement 19I21039A.

References

Alshareef A., Berthier N., Schewe S., and Huang X. (2023). *Weight-based Semantic Testing Approach for Deep Neural Networks*. In: The IJCAI-2023 AISafety and SafeRL Joint Workshop.

Ashmore R., Calinescu R., and Paterson C. (2021). *Assuring the machine learning lifecycle: Desiderata, methods, and challenges*. ACM Computing Surveys (CSUR), 54(5), 1-39.

Beyene T. A., and Sahu A. (2020). *Rule-based safety evidence for neural networks*. In: Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39 (pp. 328-335). Springer International Publishing.

Chen L. C., Zhu Y., Papandreou G., Schroff F., and Adam H. (2018). *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. In: Proceedings of the European Conference on Computer Vision (ECCV) (pp. 801-818).

Cheng C. H., Huang C. H., and Nührenberg G. (2018a). *nn-dependability-kit: Engineering neural networks for safety-critical systems*. arXiv preprint arXiv:1811.06746.

Cheng C. H., Huang C. H., and Yasuoka H. (2018b). *Quantitative projection coverage for testing ML-enabled autonomous systems*. In: *Automated Technology for Verification and Analysis: 16th International Symposium*, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings 16 (pp. 126-142). Springer International Publishing.

Dilmegani C. (2024). *Synthetic Data vs Real Data: Benefits, Challenges in 2024*. https://research.aimultiple.com/synthetic-data-vs-real-data/#some-challenges-with-using-synthetic-data-against-real-data. Accessed 20th July 2024.

Dola S., McDaniel R., Dwyer M. B., and Soffa M. L. (2024). *CIT4DNN: Generating Diverse and Rare Inputs for Neural Networks Using Latent Space Combinatorial Testing*. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (pp. 1-13).

Goodfellow I. J., Shlens J., and Szegedy C. (2015). *Explaining and Harnessing Adversarial Examples*. arXiv preprint arXiv:1412.6572.

Hartjen L. (2023). *Semantic Classification of Urban Traffic Scenarios for the Validation of Automated Driving Systems*. Doctoral Dissertation, Technische Universität Carolo-Wilhelmina zu Braunschweig, “TU Braunschweig”.

Hawkins R., Paterson C., Picardi C., Jia Y., Calinescu R., and Habli, I. (2021). *Guidance on the Assurance of Machine Learning in Autonomous Systems (AMLAS*). arXiv preprint arXiv:2102.01564.

Hirschle M., Kirov D., Aievola R., Sinisi S., Iovino S., and Adamy J. (2023). *Scenario-Based Methods for Machine Learning Assurance.* In: 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC) (pp. 1-10). IEEE.

ISO 21448. (2022). *Road vehicles — Safety of the intended functionality*. ISO 21448:2022, 1st Edition, International Standards Organization, Geneva.

Katz G., Barrett C., Dill D. L., Julian K., and Kochenderfer M. J. (2017). *Reluplex: An efficient SMT solver for verifying deep neural networks*. In: Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30 (pp. 97-117). Springer International Publishing.

Kelly J, H., Dan S, V., John J, C., & Leanna K, R. (2001). A Practical Tutorial on Modified Condition/Decision Coverage. Technical Report TM-2001-210876. NASA Langley Technical y Research Center. https://dl.acm.org/doi/pdf/10.5555/886632. Accessed 20th July 2024.

Kim J., Feldt R., and Yoo S. (2023). *Evaluating surprise adequacy for deep learning system testing*. ACM Transactions on Software Engineering and Methodology, 32(2), 1-29.

Kolb N., Hauer F., and Pretschner A. (2021). *Fitness function templates for testing automated and autonomous driving systems in intersection scenarios*. In: 2021 IEEE International Intelligent Transportation Systems Conference (ITSC) (pp. 217-222). IEEE.

Ma L., Juefei-Xu F., Zhang F., Sun J., Xue M., Li B., Chen C., Su T., Li L., Liu Y., Zhao J., and Wang Y. (2018a). *Deepgauge: Multi-granularity testing criteria for deep learning systems*. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (pp. 120-131).

Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., Wang, Y. (2018b). Deepmutation: Mutation testing of deep learning systems. In 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE) (pp. 100-111). IEEE.

Mani S., Sankaran A., Tamilselvam S., and Sethi A. (2019). *Coverage Testing of Deep Learning Models using Dataset Characterization*. arXiv preprint arXiv:1911.07309.

Montavon G., Binder A., Lapuschkin S., Samek W., and Müller K. R. (2019). *Layer-Wise Relevance Propagation: An Overview*. In: Samek W., Montavon G., Vedaldi A., Hansen L., and Müller K. R. (eds). *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Lecture Notes in Computer Science, Vol 11700, pp 193-209. Springer, Cham. https://doi.org/10.1007/978-3-030-28954-6_10. Accessed 20th July 2024.

Padilla R., Netto S. L., and Da Silva E. A. (2020). *A survey on performance metrics for object-detection algorithms*. In: 2020 International Conference on Systems, Signals and Image Processing (IWSSIP) (pp. 237-242). IEEE.

Pagano T. P., Loureiro R. B., Lisboa F. V., Peixoto R. M., Guimarães G. A., Cruz G. O., R., Araujo M. M., Santos L. L., Cruz M. A. S., Oliveira E. L. S., Winkler I., and Nascimento E. G. (2023). *Bias and unfairness in machine learning models: a systematic review on datasets, tools, fairness metrics, and identification and mitigation methods*. Big Data and Cognitive Computing, 7(1), 15.

Pretschner A. (2015). *Defect-Based Testing*. Dependable Software Systems Engineering, 84.

Redmon J., and Farhadi A. (2018). *Yolov3: An incremental improvement*. arXiv preprint arXiv:1804.02767.

SAE. J3016. (2021). *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Standard J3016_202104, 30th April 2021. SAE International, Warrendale, PA.

Stallkamp J., Schlipsing M., Salmen J., and Igel C. (2011). *The German traffic sign recognition benchmark: a multi-class classification competition*. In: The 2011 International Joint Conference on Neural Networks (pp. 1453-1460). IEEE.

Sun Y., Huang X., Kroening D., Sharp J., Hill M., and Ashmore R. (2018a). *Testing Deep Neural Networks*. arXiv preprint arXiv:1803.04792.

Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., & Kroening, D. (2018b). Concolic testing for deep neural networks. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (pp. 109-119).

Tambon F., Laberge G., An L., Nikanjam A., Mindom P. S. N., Pequignot Y., Khomh F., Antoniol G., Merlo E., and Laviolette F. (2022). *How to Certify Machine Learning Based Safety-critical Systems? A Systematic Literature Review*. Automated Software Engineering, 29(2), 38.

Willers O., Sudholt S., Raafatnia S., and Abrecht S. (2020). *Safety concerns and mitigation approaches regarding the use of deep learning in safety-critical perception tasks*. In: Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39 (pp. 336-350). Springer International Publishing.

Zendel O., Murschitz M., Humenberger M., and Herzner W. (2015). *CV-HAZOP: Introducing Test Data Validation for Computer Vision*. In: Proceedings of the IEEE International Conference on Computer Vision (pp. 2066-2074).

Zendel O., Murschitz M., Zeilinger M., Steininger D., Abbasi S., and Beleznai C. (2019). *RailSem19: A Dataset for Semantic Rail Scene Understanding*. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops.

Zhao X., Huang W., Bharti V., Dong Y., Cox V., Banks A., Wang S., Schewe S., and Huang X. (2021*). Reliability Assessment and Safety Arguments for Machine Learning Components in System Assurance*. arXiv e-prints, arXiv-2112.

Zohdinasab T., Riccio V., Gambi A., and Tonella P. (2021). *Deephyperion: Exploring the Feature Space of Deep Learning-Based Systems through Illumination Search*. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 79-90).

This collation page left blank intentionally.