Testing of Machine Learning Systems: A Comprised Workflow

Lennart Franck
Hochschule Flensburg
University of Applied Sciences
lennart.franck@stud.hs-flensburg.de

***Abstract*—Scientific progress in the field of artificial intelligence has uncovered previously unattainable potential in the machine execution of highly complex tasks. Autonomous systems such as machine learning are already surpassing humans in the accuracy of their predictions, which is why business and everyday processes are increasingly reliant on AI to improve their performance. However, this also increases the demand for security and reliability of such systems. Conventional software engineering practices, especially in testing, can hardly be applied to machine learning because the complex structures of data, framework, and algorithm make systematic error analysis difficult. The Oracle problem also makes it difficult to precisely define the desired results in advance, given the explorative nature of machine learning. Therefore, this study provides an overview of the what, where, why and how issues of machine learning. And, based on this, develops a process that will serve as a guide for the field of evaluation. A case which will demonstrate the process will be presented.**

***Index Terms*—Machine Learning, Software Testing, Software Engineering, Artificial Intelligence. Neural Networks, Deep Learning, DeepXplore.**

I. Introduction

Machine learning (ML) is having an ever greater impact on the real world. ML systems are often used in safety-critical environments such as autonomous driving or medical treatments. Occasionally, the systems reveal errors, which in the worst case can lead to catastrophic failures. Recent events showed that in some situations errors in ML systems can lead to death, injuries, or some other harm. In 2018 for example, an autonomous Uber car noted the pedestrian crossing the street, but not reacted immediately, which resulted in the death of the pedestrian [1]. To be fair, the accident happened in 2018 and a lot has changed in the world of machine learning and autonomous driving since then. Nevertheless, due to accidents like this, ML systems must, of course, pass reliability and accuracy tests to be published. Testing and monitoring an ML system is therefore an essential task to increase reliability, reduce technical debt and lower the long-term maintenance costs [2]. But also to test the functional requirements and non-functional requirements of a system [3].

Compared to a traditional software system, testing ML-based systems is more difficult and offers more challenges to developers. Normally, software systems are behaving deterministic by executing pre-programmed code [4]. So, the behaviour for one input will always be identical. In contrast, ML-based systems are non-deterministic, which means that the same constant test inputs can produce different outputs. This is because ML-based systems work in a probabilistic manner. The decision logic in ML systems is inferred by a training procedure from training data and can evolve [3]. Another very challenging aspect for testers of ML-based systems is to break the systems into smaller components, like units, and test them isolated from the rest. But ML systems cant be isolated like traditional software components because to detect effects in one component you need to see the system as a whole. Low accuracy, for example, of the ML model is often a correlation between the data and the framework/library [3].

One challenge of ML-based systems is the *Oracle Problem*. Testing a system involves evaluating the behaviour of the system, which is necessary to discover potential bugs or faults. For this, we need a test oracle, which sets apart correct and incorrect behaviours of a System Under Test (SUT) [5]. As Zhang et al. [3] said, “Machine learning systems are difficult to test because they are designed to provide an answer to a question for which no previous answer exists”. That’s why it is so hard to find an oracle for model verification because there is no right answer. But on the other hand, that makes sense because if there were a right answer, it would make writing ML systems obsolete [6]. The Oracle problem thus plays a major role in

the difficulties many software testers have in adapting familiar verification processes to the realities of machine learning.

This study aims to provide a systematic overview of best practices in testing machine learning systems based on the existing literature. The division into the phases: evaluation, deployment, and monitoring is based on a study by Amershi et al. [7]. During a study at Microsoft, a nine-part workflow was developed for the development process of ML systems, the late phases of which will be used in this paper. In the evaluation phase, the trained model is checked for various metrics and, if the results are good enough, it is then delivered, after which continuous monitoring tests the system for errors or deviations. To enable a standardized procedure for testing ML systems in the future, a process was developed that is intended to serve as an orientation for testing classification and regression tasks in supervised learning.

First, in Section II to V will be the theoretical background of model testing, deployment and monitoring, which will be necessary for more in-depth understanding. The developed process is presented in section VI. Subsequently, the effectiveness of the developed guide is evaluated in a practical context thanks to a case study. Possible limitations and optimization possibilities of the model are discussed in section VII. Finally, the results of the study are summarized, and future research approaches are discussed in section VIII.

As mentioned before, the goal of the paper is to give a better understanding of the topics model evaluation, model testing, and model presenting. But also to instruct in the form of a workflow that can be applied to any ML-based system for testing and evaluation.

II. Background

In this section, I will be providing some background information in the form of key terms and concepts about Machine Learning (ML) programs in general and testing of ML-based systems. We also discuss *what*, *why* and *where* to test ML programs.

A. Key Terms

This terminology is based on Zhang et al. [3] key terminology.

Dataset: a collection of data that is used to build or evaluate a machine learning model. The data is categorized in training data, validation data and test data.

Instance: a piece of data recording the information about an object.

Feature: an individual measurable property or characteristic of a phenomenon. Each piece of information included in the representation of an object is known as a feature [8].

Label: a value or category assigned to each data instance.

Framework: a library, or a platform, that is used to build the machine learning model. Popular frameworks include *Pytorch, TensorFlow, Scikit-learn, Theano, Caffe* and *Keras.*

Test error: the difference of a trained model on a set of data that it has previously never been exposed to the trained model.

Generalisation error: the expected difference between real conditions and predicted conditions in form of any valid data.

Model: processes data to make predictions. The model is trained from training data, the framework, and the learning program.

Machine learning approaches:

1) **Supervised learning:** the approach uses training data *(x, y)* with their desired outputs. The goal is to produce a prediction y^* responding to a query x^* [9]. It is the most common approach.
2) **Unsupervised learning:** the used training data does not contain labels, and therefore the learning algorithm itself must find understanding in its inputs.
3) **Reinforcement learning:** the data comes from a dynamic source in the form of sequences of actions, observations, and rewards. As it is moving forward in the problem space, the learning algorithm is provided with feedback. The goal is to maximize the positive feedback.

Machine learning tasks:

1) **Classification:** the task is to assign a category to each data instance. Examples in this space are handwriting recognition, spam classification or image classification.
2) **Regression:** predicting a value for each data instance. Examples are temperature prediction and stock prediction.
3) **Clustering:** portioning the data instances into homogenous groups. Examples are pattern recognition or community recognition of social networks.
4) **Dimension reduction:** the task is to transform the high-dimensional data into low-dimensional data. The goal is to lower the training complexity. Examples are speech recognition or bioinformatics.

5) **Control:** the goal to control actions is to get a maximized reward. Examples are game playing.

Figure 1 shows a brief overview of machine learning approaches and the corresponding tasks.

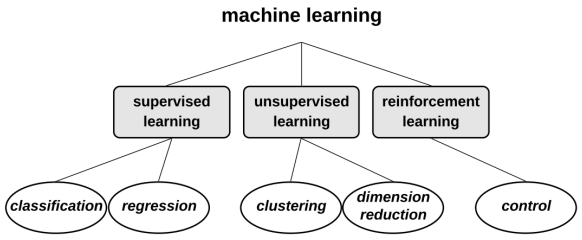


Figure 1: Machine learning categories and tasks [3]

B. Key Concepts

This subsection will give some definitions and information about the key concepts of machine learning. It will describe **how**, **why**, **where** and **what** to test in machine learning applications [3]. To get a quick overview of the following topics, figure 2 gives a quick overview of the key concepts of testing in machine learning.

Testing properties. In testing machine learning systems, there should always be a *why* to test machine learning. The requirements for testing are called testing properties. These testing properties get classified in functional requirements which involves correctness and model relevance and non-functional requirements that involve efficiency, robustness, fairness, security, and reliability.

Testing components. A machine learning software developer often is confronted with the same activities for each project. Collecting data, prepare data if necessary, choose an algorithm and framework is often a required task in every ML application. Therefore, there are numerous places where bugs and failures can occur. That's why each of these components must be tested. The problem is to identify bugs, when each component is interacting with each other. Compared to traditional software testing, the search for bugs is far more complicated because components are closely coupled with each other and therefore are harder to test individually. In traditional software development, you try to lower the coupling of an application as much as possibly. In ML application, that is often not possible because of the underlying architecture. Bug detection in data is important because ML systems depend on data and therefore the data needs to be bug free. As Breck et al. [10] identified, the bugs in data can not only affect model quality but more

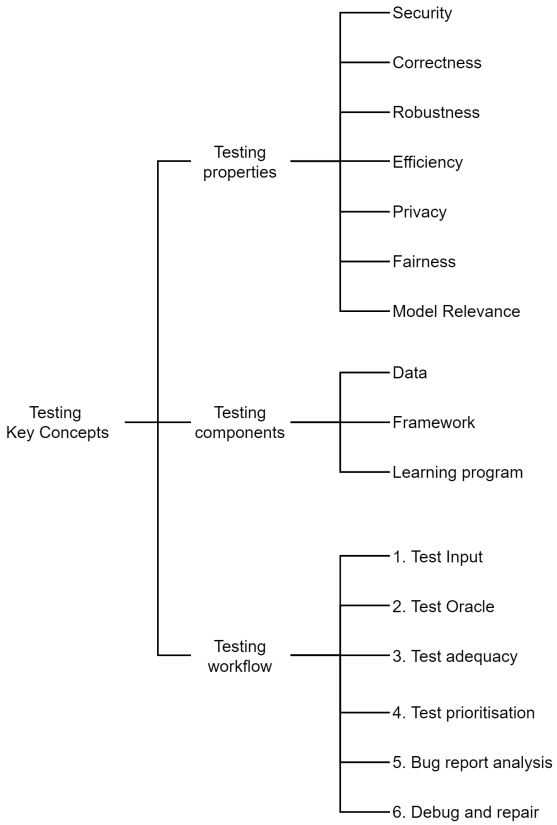


Figure 2: Quick Overview of Key Concepts of ML Testing

problems over time considering feedback loops. So, data bug detection contains checking the completeness of data, the representativeness of data, the noise of data and the distortion between training and test data [11]. Frameworks in machine learning applications have a far higher importance than compared to traditional software development. They help you write a learning program and test the model with easy to use solutions. So, it is obvious that bugs in the framework are going to have an impact on your model later on. The learning program can also have bugs, namely in the algorithm which is self-developed or chosen from a framework and the code which implements and configures the algorithm.

Testing workflow. The testing workflow is about how to do ML testing. This includes different testing activities. This is a short representation of how a workflow could be. In section VI I will be giving a fully detailed version of a workflow as a solution when dealing with ML testing. A testing workflow should at least include the steps of test input generation, test oracle generation, test adequacy, test prioritization, bug report analysis and

debug and repair.

III. Evaluation

In the following section there will be analysis of different usages of testing regarding the following aspects: *Test Input Generation*, *Oracle Testing* and *Test adequacy*.

A. Test Input Generation

To be able to evaluate the result of machine learning systems such as neural networks and deep learning networks, the input data of the model, i.e., the training and test data, should be evaluated. This is important because the ML algorithm uses this data to create patterns to generate its predictions. If this data do not exhibit normality, i.e., are not representative of the entirety of the data sets to be processed, the algorithm cannot generate accurate forecasts for new, unknown data.

Many statistical test procedures can be found within science for the evaluation of input data. For example, one could use the *Chi-Quadrant-Test* to look up dependencies between variables in the data. Another test is the *t-Test,* which takes samples from the data and tests these on their mean difference. There are different variants of the test depending on the number of samples and dependencies with each other.

Test Inputs can be classified into *adversarial inputs* and *natural inputs*. We will focus on the latter of these because natural inputs belong to the data distribution of a practical application scenario.

DeepXplore [12] is a white-box Framework made by Pei et al. to generate test inputs for deep learning systems. DeepXplore uses neuron coverage, inspired by test coverage which is normally used in traditional software testing, to create test inputs that have a high neuron coverage. This is necessary to build different corner cases to detect and fix ideally any potential flaws or undesired behaviours [12]. DeepXplore wants to tackle two challenges for large automatic systems:

- How to generate inputs that trigger different parts of a deep learning system's logic and uncover different types of erroneous behaviours
- How to identify erroneous behaviours of a deep learning system without manual labelling checking

Tian et al. [13] created *DeepTest* which is a systematic testing tooling for Deep Learning (DL)-driven vehicles. This makes sense because autonomous driving is an extremely sensitive area because of the high consequences, errors may have. DeepTest is designed to automatically generated test cases leveraging real-world changes in driving conditions like rain, fog, lighting conditions, etc. But it also, like DeepXplore, systematically explores parts of the DL logic by generating test inputs that maximize the numbers of activated neurons [13]. DeepTest detected more than 1000 erroneous behaviours on CNNs and RNNs with low false positive rates[1] [3].

Zhou et al. [14] had the same idea with DeepBillboard. They generated real-world Billboards that could potentially cause errors like false steering of a car.

B. Oracle Testing

To address the oracle problem, we look at metamorphic relations testing. An additional method for evaluating machine learning systems is so-called metamorphic relation testing. Here, the input data is systematically transformed and then passed back to the ML program to observe the change in the results. "A metamorphic relation refers to the relationship between the software input change and output change during multiple program executions" [3]. Metamorphic testing also addresses the Oracle problem, which describes the difficulty that the correct result for the respective input data of an ML algorithm is not known in advance [5]. For this reason, ML programs are often said to be impossible to test. Metamorphic relations are transformations on training or test data that expect to give certain changes in the predictive output [3].

As an example for metamorphic testing Tian et al. developed the DeepTest program, which modified the input data for the classification of images in autonomous driving in such a way that fog and rain of varying intensity were added to the original input data. This increased the accuracy of the tested system [13]. A link between testing the behaviour and the neuron coverage of the system is provided by DeepXplore, presented by Pei et al. Through this program, new input data is generated, which should cause maximum coverage of the neurons of the deep learning system concerned. Neuron coverage is the calculated ratio of the number of all activated individual neurons by the amount of all test inputs to the total number of neurons in a deep neural network. The goal is to optimize the test data set. A neuron is activated when the output value is greater than the user-defined threshold. It is therefore possible that the neurons of many layers of the neural network are not activated at all and the system consequently delivers an inaccurate, error-prone result. It is an innovation in the optimization of Neuron Coverage.

[1] https://deeplearningtest.github.io/deepTest/

C. Test adequacy

Adequacy testing is to find out whether the test someone used are good enough regarding fault-revealing. The two most popular test adequacy evaluations are *code coverage* and *mutation testing*. They are adopted by traditional software testing.

We talked about Neuron Coverage before when talking about DeepXplore [12]. This is an alternative compared to in traditional software testing to test code coverage. It's specifically for deep learning testing but is the first really conception in ML Testing of a coverage criterion. In addition to that, Sun et al. [15] proposed four test coverage criteria that are inspired by the MC/DC coverage criteria. The MC/DC test procedure observes the change in Boolean variables. It is examined whether there is a change in the sign, value or distance of a neuron. In this way, the following necessary changes in the test inputs are to be recorded. Layer-level coverage was introduced with *DeepGauge* by Ma et al. [16]. "At layer-level, we use the top hyperactive neurons and their combinations (or the sequences) to characterize the behaviors of a DNN" [16]. The result was that, together with neuron coverage, there was a better performance on the datasets MNIST and ImageNet.

Mutation testing in traditional software testing is about injecting faults into a system to detect the fault revealing abilities of the testing suite [3]. The ratio of detected faults versus injected faults is then called the *Mutation Score*. In the introduction to this paper (section I) I talked about how in ML systems all components depend on each other like the data, the model, the model structure and the algorithm etc. With *DeepMutation* Ma et al. [17] created a framework to to inject faults to the source of deep learning (training data and training programs). Then design a set of model-level mutation operators that directly inject faults into DL models without a training process. Eventually, based on these injections a mutations score, defined as the ration of test instances of which results are changed against the total number of instances, is given [3, 17].

IV. Deployment

The deployment of machine learning applications is one big step to publish your application to real-world data. Now the application comes into the Online-Testing phase. But before the machine learning system can be published, it is first checked whether the requirements for the application are all fulfilled after the offline tests [3]. Cross-validation and other before discussed methods (neuron coverage) can be used to check the system properties (correctness, robustness, etc). If everything is giving correct projections, the next step is the deployment, i.e. the publication of the developed system.

A short overview of the deployment of a machine learning application is shown in figure 3. First, there is a model which is trained by static training data and tested by static test data. After training and testing, the model makes predictions. After that, there is production-ready code, if the following steps happened:

1) The code should include clear instructions and clean code. This includes easy to understand functions and variable names, functions with only one task and a documentation and type usage.
2) The code you deploy should be tested. This gives confidence that your code functions run as expected and that when you make changes, you didn't actually break anything. More on this in the testing workflow.

With these steps completed, there is another step to take before you can deploy your code. The code needs to be containerized. This could be realized with the Docker software, for example. Docker is explaining the container system as the following: "A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings" [18]. The advantage: New images can be easily created and published. Existing images can be updated, replaced or downloaded for archiving. In the Docker Registry, images can be stored publicly and also privately for backup purposes [19].

After the Code is containerized, it is ready to be deployed in the cloud. But before that, one should consider how to access the model and get predictions from the model. The easiest way this can be done is by putting the model behind an Application Program Interface (API) which is then exposed to a webpage, mobile apps, etc. The API would take in the necessary data for the model, normally in JSON format, to make predictions and then return the prediction based on the data. Because most ML developers work in Python, a recommendation for a Python-based API would be Flask or FastAPI.

Now the code which contains an API and the model can be deployed to your favourite cloud provider. The three main providers are AWS, Google Cloud and Microsoft Azure. All have the similar functionalities and

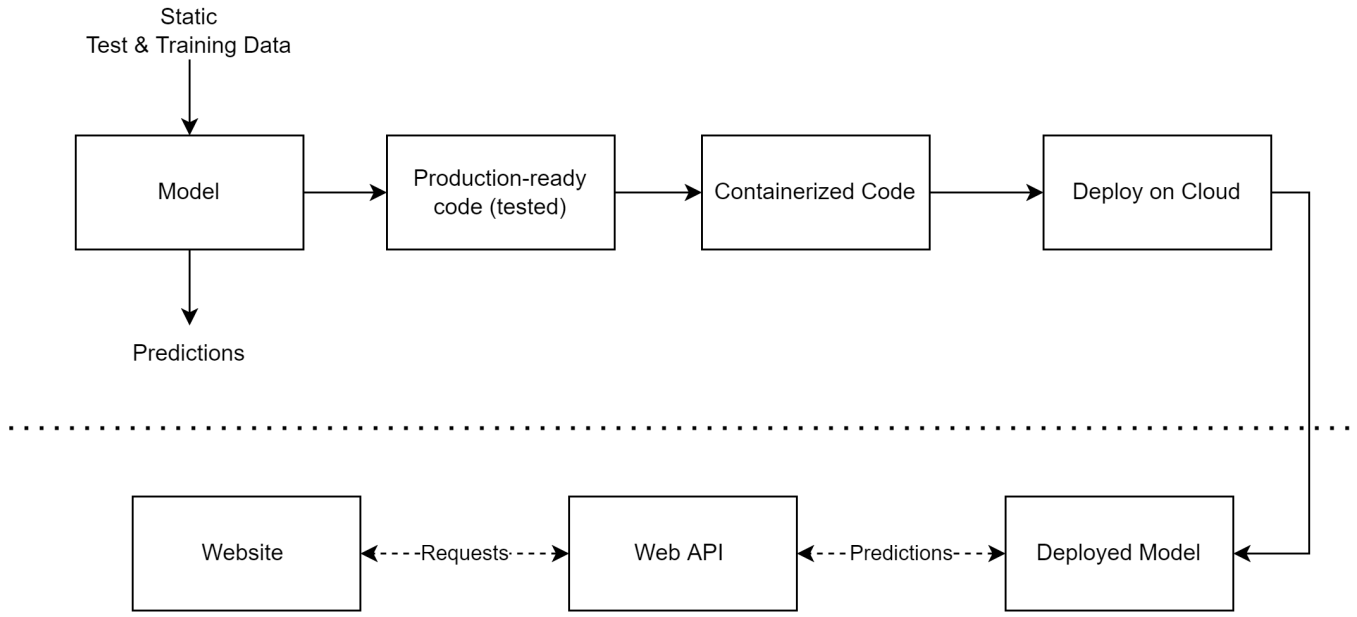


Figure 3: Deployment of Machine Learning Models

only have small differences in pricing. But there are also smaller cloud provider which could be promising to use.

Of course this is a very oversimplified presentation of the whole process and there is almost infinite complexity when it comes to ML deployment, but this should just give a short overview.

A. *Online Testing*

Because the deployed model should get more in-depth understanding of real life data, the model should also be tested online. Online tests are conducted because there are components that cannot be tested before the product is made available to the user. Offline tests are based on test data from the past. This cannot represent the future data in all its complexity. For this reason, new data is sent to the system after deployment, which is then intended to improve correctness. Tests like delay on predictions and data loss cannot be tested locally. Developers can use beta-testing to ensure a production-ready applications runs as expected. A/B testing is one other type of online testing. This is used more frequently later in the monitoring process. This test procedure involves comparing two possible versions of the product with each other, evaluating them in terms of user behaviour, and further developing the better version.

V. Monitoring

In this section, I will take a short insight into an important, but typically underestimated, step. Monitoring the production model. To monitor the performance of neural networks in use, A/B testing can be carried out or various visualization options can be applied. The monitoring should observe the properties of the data and their relevance. In addition, the resource consumption of the application should be monitored and compared to the generated benefit, as should the costs incurred. The time factor should also be considered appropriately in this phase by regularly monitoring the time required to provide a model.

A. *Monitoring Types*

Runtime Monitoring can be used to check if the ML systems meet the desired requirements and correction or some failures is detected. Furthermore, it can be used to log unexpected behaviour or automatically trigger corrective actions like shutting down an autopilot [20]

A/B-Testing is a very typical test for online testing, but it can also be used to monitor ML systems. It's done by splitting a system into two versions that involves user interactions. The aim of an A/B test is to decide between two alternatives that differ from each other in only one aspect.

Multi-Rrmed Bandit (MRB) is an online testing approach that is similar to A/B-Testing but with this, further resources are added only to the chosen selected model [21].

VI. Solution

In the following section, I will finally give an insight of the workflow that is best on my experience for testing machine learning applications. Of course, new discoveries within testing ML applications can occur that would make some steps obsolete. ML and ML testing is an ever-growing space, and inventions will come now or later.

The full graphical workflow is shown in figure 4. After the feature engineer has handed over the model and the input data, the data received should first be evaluated and checked for suitability and representativeness. For this, one can choose from a variety of statistical tests. The chi-square test is used to test the dependence of individual variables on each other. Or the t-Test, which takes samples from the data and is testing these on their mean difference. If the tests measure the dataset as not sufficiently representative or suitable enough for testing and training the machine learning system, the tester should return the input data to the feature engineer and ask for correction. If, on the other hand, the data is found to be well-grounded to this point, the data is further investigated using the Test Input Generation and Test Oracle Generation.

With Test Input Generation, the goal is to find corner cases that identify undesired behaviour or potentially bugs in a system. To achieve that, DeepXplore [12] for example created Test Inputs using neuron coverage. It explores parts of a system by maximizing the numbers of activated neurons. To be fair, this process only works good for DL. With Test Oracle Generation, we look at metamorphic relations testing, for example. Metamorphic testing measures the accuracy of forecasts in response to modified input data. In this way, the robustness of the system is tested. The change in the output is observed and tested for correct behaviour. At this point, the outputs can also be analysed regarding surprisingly occurring input data. Data that deviates strongly from the average of the training data is distributed into the ML system. If the deviations from the desired result are too great, the input data are returned to the feature engineer. When the desired results are achieved, the structure of the system is examined.

In a machine learning system, for example, the structure consists of a trained DNN. Depending on the size of the data sets and the possible output data, DNN have different numbers of layers and consequently also different numbers of neurons. When testing the structure thanks to programs such as DeepXplore, it is found out whether the neural network is used sufficiently enough. Test procedures to determine the neuron and layer coverage show whether certain paths may not be used at all or only insufficiently. If this is the case, the feature engineer must check the design and structure of his system and, if necessary, change it. If the system is used sufficiently, we can analyse the overall quality of the system.

Quality Testing means the testing of the important properties like correctness, robustness, or security. How these properties are to be measured is very dependent on the system and architecture one is using. For example, in the case of classification, many possible metrics are available. To ensure the correctness of the system, a confusion matrix with the key figures Precision, Recall and F-Score can be set up and checked. Robustness, on the other hand, is a non-functional characteristic of machine learning systems [3]. Therefore, the robustness can be checked by the correctness of a system with existence of noise. Perturbations (added noise) that fool the system can also quantify the robustness.

Then the requirements for the machine learning system are checked one last time before the application is finally published. The deployment workflow in figure 3 should give some insight into the deployment of an ML system. Software applications like Docker and Cloud Provider like AWS or Azure give some great capabilities for publishing an application. After the application has been published, possibly only for testing purposes at first, the system is tested in the online environment. Online testing offers the tester the collection of metrics and behavioural patterns that cannot be captured offline. For example, factors such as click frequency, opening rates or reading time can now be measured. Another thing you can do in online testing is A/B Testing with two variants of the model. But the difference between the systems should just be a minor one. After A/B Testing, you can decide which version performed better based on some metrics and take the model to the monitoring stage.

In the monitoring stage, you monitor the predictions of a system and also monitor the input data. Because it is important to know not just that your ML system worked correctly at launch, but that it continues to work correctly over time [2]. ”The standard approach is to monitor the system, i.e., to have a constantly updated “dashboard” user interface displaying relevant graphs and statistics, and to automatically alert the engineering team when particular metrics deviate significantly from expectations” [2].

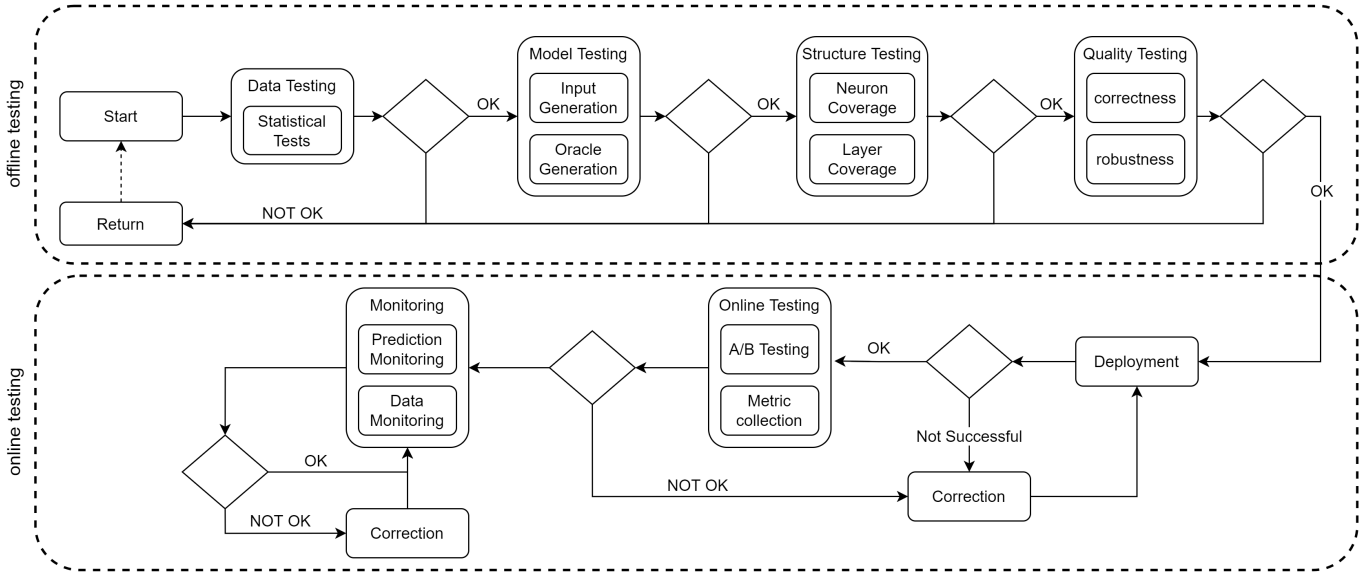


Figure 4: Workflow for Machine Learning Testing

VII. Case

In the following chapter, we will have a closer look into what a real example of the workflow in figure 4 could look like. For that, we use a Classification Problem. I decided to choose one of the most well-known datasets, the MNIST dataset. MNIST is a large database that contain handwritten digits in the form of 28×28 pixel images [22]. The images are labelled with digits from 0 to 9. The complete dataset consists of 60,000 training and 10,000 testing samples. For the model, I decided on a convolutional neural network (CNN) based on LeCun et al. [23], the LeNet-1,4 and 5. CNNs are a kind of multi-layer neural networks. Like every other neural network, they are trained with a version of the back-propagation algorithm. But the special thing about CNN is they are designed to recognize visual patterns directly from pixel images with minimal preprocessing. They can recognize patterns with extreme variability and with robustness to distortions and simple geometric transformations [23]. For example, the LeNet-5 architecture contains 5 Layers which have activation functions and sometimes pooling for reshaping between them.

Because of limitations in the research and development of this paper, I will not be showing a full case demonstration. I will give insight into one testing aspect to give an example how the workflow can be used. Because of the chosen MNIST dataset, I will be ignoring the data testing part because I know the data is in a correct format and therefore there is no need to test that. Performing statistical tests on image data is also not very promising.

I will test the model with the input test generation and neuron coverage. For this, the DeepXplore algorithm created by Pei et al. [12] will be used. DeepXplore uses neuron coverage to create test inputs that have strong activations on the model. That creates corner cases for testing ML models. DeepXplore uses three different types of constraints for simulating different environment conditions of images [12]:

1) Lightning effect for simulating different intensities of light
2) Occlusion by a single small rectangle for simulating something that is blocking the camera
3) Occlusion by multiple tiny black rectangles for simulating dirt on the camera

The DeepXplore algorithm is defined as a joint optimization problem with the goal to maximize both the number of observed differential behaviours and the neuron coverage. DeepXplore solves this problem by using the gradient ascent method. Differential behaviours mean that for n DNNs tested where x is representing the input and y is representing the output, the goal is to modify x so that is at least one of the n DNNs will have a different outcome y. To maximize the neuron coverage, the algorithm is iteratively picking inactivated neurons and modifying the input so that the output goes above the neuron threshold.

It does these actions by comparing three or more models. It loops through the data and generates new images with a new axis that will appear at the axis position in

the expanded array shape. If one of the models gets a different prediction on the image, the neuron coverage of all models gets updated. If the models all have the same prediction, the image will be changed with one of the environment conditions named above. If the prediction on the newly created image is not the same for all three models, then the image is a corner case image. This means that the coverage gets updated again, and the image will be saved. The saved images can be used to again train the model and maybe get a better error loss because of the new corner cases images. Because the implementation is quite complex, I decided to take an example from someone who implemented the algorithm. The code itself is from a GitHub Repository[2] which took the original DeepXplore code base[3] and migrated it into Tensorflow 2.0 and retrained the model with the images created by the DeepXplore algorithm.

A. Discussion

The aim of the case study was actually to create a comprehensive evaluation of the feasibility and precision of the developed guideline for the systematic testing of an exemplary practical case. The MNIST dataset and the LeNet Convolutional Neural Networks were selected for this purpose. However, the goal of a complete implementation failed, so that alternatively a partial aspect of the process were shown as an example of possible implementation scenarios. An actual execution of comprehensive tests was not possible because the scope of the necessary literature research for the basic understanding of the numerous tests and complex algorithms exceeded the original time planning. An earlier completion of the literature review would thus have been absolutely desirable, but was not possible due to the low familiarity with the complex practices in ML testing in the context of this study.

VIII. Conclusion

The attempt to reproduce human intelligence in its completeness has always occupied science. With the development of machine learning systems such as neural networks, enormous progress has been made recently, which now makes it possible to solve highly complex problems through independent learning of ML applications.

In the modern world, ML applications are everywhere, starting with the spam detection in your mailing software and ending with fully self-driving cars. But because there these risky applications in sectors like traffic where people could potentially die from failures of others, the safety of such systems is essential. The public's concerns about ceding their safety and decision-making power to machine processes through innovative concepts such as autonomous driving require a comprehensive test concept for machine learning systems to eliminate risks and ensure the robustness of the software even in exceptional situations.

But testing ML applications is not easy and can even challenge a professional and experienced testing developer. This is because ML systems differ so much from traditional software systems. Proven methods and concepts cannot be applied to ML systems because of how ML works. The Oracle problem as a lack of knowledge around the expected results of the learning processes and the relation of the components, which can be strongly influenced by the quality and type of data used, causes the need for new testing methods to verify the functionality of ML systems.

First, this study provided an overview of new testing concepts and existing best practices in ML research. To give some background knowledge about machine learning in general, I created key terms and key concepts that were based on Zhang et al. [3]. Based on a study by Amershi et al. [7], the phases' evaluation, deployment, and monitoring of a machine learning model were then selected as the basic structure of this paper. A process was then developed to guide the systematic implementation of tests in ML.

A planned full-scale case study, could not be realized because the high effort in literature search and analysis made it impossible to prepare for the complex algorithms for the selected dataset. Since the studies examined very individual problems and provided little information about generally applicable rules for ML testing or the possibilities of applying them, newcomers in testing cannot expect detailed support in deciding between alternative methods and in evaluating individual tests. Every problem and every solution to a problem is different, that's also the case for testing these solutions in the sector ML. You cannot provide a test that is compatible with all types of algorithms and data inputs.

My lack of familiarity with the specifics of ML testing made writing this paper difficult. A basic literature review beforehand was necessary to understand the basics of all published papers in this area. But I hope this paper and the distributions, including the constructed workflow, can also give beginners insight into the trends, directions,

[2]https://github.com/ee17b031-iittp/MNIST-Retrain-with-DeepXplore

[3]https://github.com/peikexin9/deepxplore

and challenges of ML testing.

REFERENCES

[1] Samuel Gibbs. Uber's self-driving car saw the pedestrian but didn't swerve – report, 2018. URL https://www.theguardian.com/technology/2018/may/08/ubers-self-driving-car-saw-the-pedestrian-but-didnt-swerve-report.

[2] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *Proceedings of IEEE Big Data*, 2017.

[3] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[4] Houssem Ben Braiek and Foutse Khomh. On testing machine learning programs. *Journal of Systems and Software*, 164:110542, 2020.

[5] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015. doi: 10.1109/TSE.2014.2372785.

[6] Martin D Davis and Elaine J Weyuker. Pseudo-oracles for non-testable programs. In *Proceedings of the ACM'81 Conference*, pages 254–257, 1981.

[7] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[10] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. Data validation for machine learning. In *MLSys*, 2019.

[11] Chih-Hong Cheng, Chung-Hao Huang, Harald Ruess, Hirotoshi Yasuoka, et al. Towards dependability metrics for neural networks. In *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pages 1–4. IEEE, 2018.

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

[13] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[14] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 347–358. IEEE, 2020.

[15] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

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

[17] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111. IEEE, 2018.

[18] What is a container?, 2022. URL https://www.docker.com/resources/what-container.

[19] Tatiana S Demidova, Anton A Sobolev, Anastasia V Demidova, and Migran N Gevorkyan. Using docker to deploy computing software. In *ITTMM (Selected Papers)*, pages 38–46, 2019.

[20] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. Model assertions for monitoring and improving ml models. *arXiv preprint arXiv:2003.01668*, 2020.

[21] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best-arm identification in multi-armed bandit models. *The Journal of Machine Learning Research*, 17(1):1–42, 2016.

[22] Yann LeCun, Corinna Cortes, and Cristopher JC Burges. The mnist database of handwritten digits,

1998. URL http://yann.lecun.com/exdb/mnist/.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.