Implementation of MLOps

Maximilian Zwiesler

June 2023

Abstract

The rise of machine learning (ML) has reshaped industries, demanding efficient ML model deployment. This has led to the emergence of Machine Learning Operations (MLOps), which streamlines the entire ML workflow. MLOps aligns data, model, and code components to ensure accurate and scalable models. However, testing and diverse execution environments are underrepresented in existing literature. This paper bridges this gap by presenting crucial tests for reliability and proposing an integrated MLOps framework that unifies testing with different environments. The paper structure encompasses foundational definitions, a standardized development process, and concludes with insights into enhancing MLOps reliability.

1 Introduction

1.1 Motivation

Machine learning (ML) has become an important part of many industries, ranging from healthcare to finance, and from retail to transportation. However, for most of them deploying ML models into production environments is relatively new. Until now there might have only been a manageable number of models which needed to be hosted. As the number of models and the role they play for the company business growths, since more and more decisions are automated, the importance of being able to develop, deploy and operate the models efficiently increases. This is where Machine Learning Operations (MLOps) comes into play. [22]

MLOps is an emerging discipline that includes the entire machine learning workflow, from data preparation and model training to deployment and monitoring. It aims to standardize and automatize the process of developing and deploying ML models in a production environment by integrating best practices from software engineering (DevOps – development and operation) and data science. [16]

At the core of MLOps there are three components: data, model, and code. These three components work together to ensure that ML models are accurate, scalable, and reliable. The

data component involves preparing and managing the data used to train and evaluate machine learning models. The model component involves developing and testing machine learning models to ensure that they meet the desired accuracy and performance requirements. The code component involves the software engineering practices used to package, deploy, and maintain machine learning models in a production environment. The integration of these three components is critical to the success of MLOps. [13]

1.2 Research Question

The principles of DevOps were first mentioned in 2007 and since then have been discussed extensively. [2], [8] Thus, the process how to write, test and deploy code is well established. [4], [7] From this approach the framework of MLOps has been developed. [30] The main focus of the existing literature is defining a suitable process how to train and deploy an ML model and the tools [10], [13], [18], [26], [36] and skills [16], [22], [30], [35] needed to implement it. Even though testing is one of the most important features of DevOps these reports only provide a high-level overview of required tests or focus on special parts of testing like the performance of the ML algorithm.

Only few articles focus on a valid testing strategy without directly linking them to specific parts in the MLOps process. [5], [21] Additionally, often the process is not considering the usage of different environments. It is generally acknowledged that the testing of MLOps is very complex and requires further investigation. [27] To close this gap this paper contributes to the literature in the following way:

The required tests to ensure a reliable process are presented and a suitable MLOps framework is proposed which describes the integration of these tests in combination with different execution environments in detail.

1.3 Structure

The paper is structured in the following way. First the general definitions of underlying principles used by MLOps are provided (chapter 2). This includes an introduction to DevOps and staging environments as well as an extensive overview over all required tests which should be executed during the MLOps process. The third chapter will define a standard development and deployment process starting with consuming the data, followed by the training of the model and finally its deployment. The workflows also consider the execution of the different steps in different environments. Especially, the various tests which were described in chapter 2 are part of the proposed processes. The last chapter concludes.

1 Underlying Theory of MLOps

In this chapter the technical principles of MLOps are described. First the concept of DevOps is introduced. This usually comes with the need of different staging environments which are explained next. Finally, the different tests are described which need to be executed to ensure a reliable process which can be automatized. The following chapters will then explain how these concepts are used in the process of MLOps and in detail for the example at hand.

1.1 DevOps

As already stated, MLOps is built upon the principles of DevOps. Thus, in order to be able to understand and apply MLOps we first need to understand the idea behind DevOps.
The focus of DevOps is to merge development, testing and deployment into one continuous operation. The goal is to speed up the whole process through automation. This ensures that Development and Operation teams can work together in an agile way, leading to faster software development, improved quality, and better customer involvement. [8]

This is achieved by mainly two concepts: [15]

- Continuous Integration (CI): This is a software development concept where team members integrate development work in small batches and thus more frequently. Here each change triggers automated software building as well as execution of tests. In case failures are detected these are fixed immediately before the changes can be promoted to the next step. This process will result in shorter releases and improve the quality since it ensures that all changes made by different developers are tested and compatible.
- Continuous Delivery (CD): Continuous delivery continues where CI left off. After successfully passing the automated tests, this step ensures that the application is delivered to the production environment. This includes packing the code (or model) into deployable artifacts and deploying these artifacts.

Although MLOps is based on DevOps there are some significant deviations, especially, since MLOps does not only depend on well-understood programming code but needs to handle data and models in addition to code in the process. Therefore, MLOps requires certain adaptions compared to DevOps. [18]
First, in contrast to software applications model training in ML is an experimental field. Thus, it requires significant efforts with respect to data preparation, model training and deployment of

the model into an application. If an error needs to be fixed one cannot simply edit the code, test, and deploy it but has to cover these additional tasks as well. [10]

Second, the testing framework consisting of unit and integration tests is key to ensure the correctness of the software. In the context of developing ML models these tests only cover one part but do not account for data or model quality checks. Figure 1 shows the increase in complexity between the two approaches. Left the approach of traditional software engineering is shown. Here only the code of the running system needs to be tested including unit and integration tests and monitoring of the productive system. The right side shows a possible testing approach of a machine learning system. Additionally, to the existing tests of the code also model, data and infrastructure needs to be tested with integration tests covering all aspects. The individual tests will be explained in more detail in the next section.

Finally, once a model is deployed it uses real-time data to make predictions. The performance of the model does not only depend on the correctness of the training but is also affected by other factors like data shifts or business goal adjustments. Hence, additional types of monitoring are required. [5]

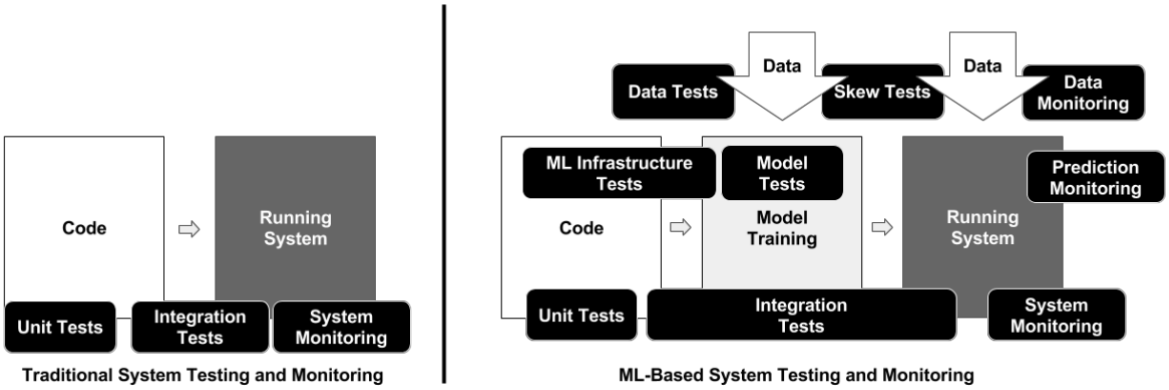

Figure 1: Different test requirements for DevOps (left), and MLOps (right) (reprinted from [5])

1.2 Staging Environments

As already stated, the MLOps process consists of three main parts: code, models, and data. All of them need to be developed (dev), tested (integration) and deployed (prod). For each of these steps there is a need to operate in a separated execution environment. Hence, all parts (code, models, data) are used in a dev, int (integration) and prod environment.

An execution environment consists of a compute instance with affiliated runtime, libraries, and jobs. Here code consumes and generates data and models for a specific purpose.

The main difference between these environments comes from quality guarantees and access control. Here dev is easily accessible with different people being able to experiment with no

guarantee of quality. However, since these experiments are separated from prod there is no risk for the business involved.

In contrast to dev, the integration and prod environment are more restrictive. The integration environment mimics the prod environment to test the behavior of code, data, and models. In the integration environment the code changes promoted from dev are tested to ensure that only high-quality code is integrated into prod. Since testing should be an automated process only few people should be granted access to this environment.

Once, the code passes the quality gate of the integration environment it is promoted to production. This environment thus ensures the highest quality with the most restrictive access concept since changes here can have a direct business impact. [13]

It is important to understand, that in the MLOps process there are two processes which call for separate environments. On the one hand the model development process which is usually executed in an analytics environment. And on the other hand, the deployment process where the model is wrapped inside an application code consuming the model and the resulting application is operated in a serving environment. Figure 2 shows this separation. To ensure that a new model can be developed at any time, a model development process with the three defined environments (dev, int and prod) is needed. At the end of the process, the model is handed over to the serving process where the model is integrated into the application code. This usually takes place independent of the model development and thus also needs separate environments for the development of the code, the testing, and the operation of the application.

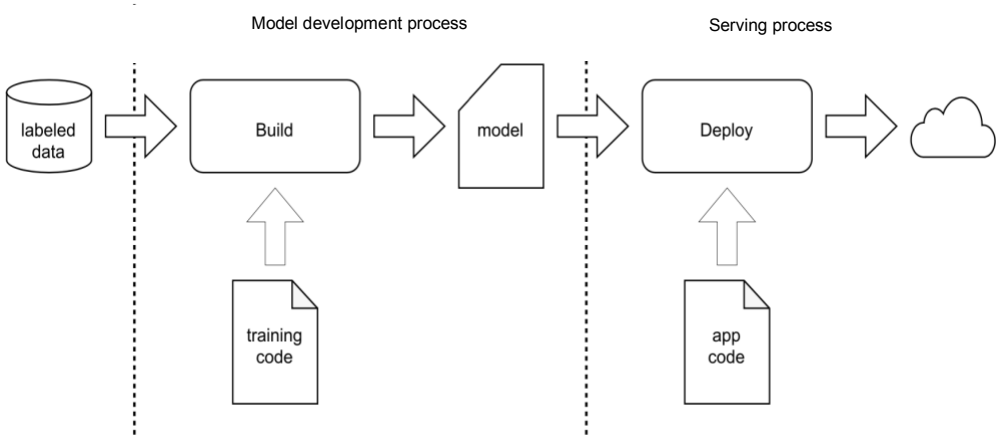

Figure 2: Functional steps for developing and deploying an ML application (reprinted from [11])

1.3 Testing

To ensure the quality of the complete MLOps process one must understand the requirements of tests. In this section an overview of the necessary tests for the complete process are listed

and described in detail. However, these are not yet linked to a specific staging environment or process step. This will be done in section 2.

As already discussed in section 1.1, there is already a common process how software applications and thus code is tested. As explained above the code for all three processes – data preparation, model development and serving – need to be tested. Additional complexity arises since code is only one part of MLOps, but the data and model components call for additional tests. Figure 3 shows a testing pyramid covering tests for the individual components and combinations for the data, the model, and the code (regarding data preparation, model development and application). A test pyramid shows the low-level tests like unit-tests at the bottom, which are very fast and cheap and thus should include a much higher number of tests, and the costly and slow high-level tests like end-to-end tests at the top. [4]

Since data plays a fundamental role in developing ML models the first pillar of testing is testing of the data. The quality of the data can be assessed by checking the completeness, accuracy, consistency, and timeliness [29]. Here one approach is to create a schema starting with determining statistics from training data which can be checked against domain knowledge. This will ensure that the input data values fall within expected ranges. Later these schemas can be used to validate new data during retraining or inference. [5]

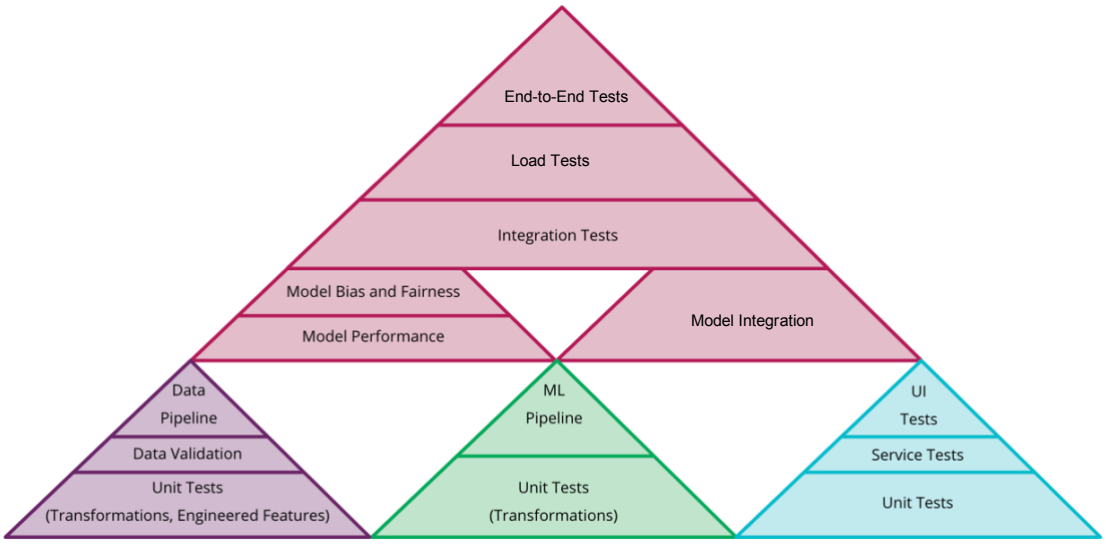

Figure 3: Testing Pyramid (reprinted from [27])

Furthermore, in case new features are engineered during the process the code to derive them should be tested as well. As an example, this includes tests that missing values are addressed appropriately. Although, this may seem to be exaggerated in case of simple feature creation, errors in this step have a huge impact on the model and are very hard to detect in the later process. [5]

The second pillar focuses on the testing of the model. First, in case of further transformations of the data within the model pipeline, e.g., due to specific need of model input like normalized values, these transformations also need to be tested. Also, the model specification code should be tested. This can be achieved by running a simple unit test, using randomly generated input data with one step of the of the training pipeline which should already provide good results. [5]
The next steps use the data to evaluate the model. These model performance tests ensure that the quality of the model is suitable. Different metrics like accuracy, AUC or confusion matrix can be used to evaluate the performance of the model and ensure a high quality. The complexity of a model should also be evaluated, e.g., by comparing it against a simple linear model. With this the cost-to-benefit tradeoff of a complex model can be estimated. [27] This will later be again taken up during the discussion of the model development process. In general, the test of model performance should cover first of all qualitative measures of the model together with further restrictions coming from the serving environment. But since the process is too complex to cover all these additional restrictions already in this step, additional tests regarding model integration are needed and will be discussed later.
Finally, the test cases should also cover proper input and output data format of the model. [30] It also makes sense to test that the model logs the required information in case model logs are necessary. Since getting insights from the model plays an important role during monitoring more complex models will need to log information about the processing of the request.
Special attention should be given to the topic of fairness and bias of a model. One should evaluate how the model behaves for specific subgroups or data slices. For example, one could have a bias in the training data regarding the gender compared to the real world. This could have severe consequences for the reputation and business of a company. [22]
The third pillar of testing considers the application testing. Regardless of whether the model is deployed as a REST API [19] or as a batch process the application itself as well as the integration of the model must be tested. In some cases, the application might not be hosted by the team developing the model. In this case the tests must be executed by the application team using the production-ready model provided by the model development team. However, a complete integration and end-to-end test is not possible in this scenario.
When integrating the model into an application it often gets exported to a different format.[1] Here one has to ensure that the productionized model still behaves the same way as the original model, by running both against the same validation dataset. To confirm that the model is compatible with the application one can deploy the model together with the application code to an integration environment. [26]

[1] See embedded model pattern in section 2.3 for details.

The integration tests ensure that the full ML Pipeline is working as expected. This usually contains the data pipeline, the data transformations, the model training, the model performance, and the deployment into an application. Since the full pipeline roughly consists of two parts, the model development, and the application part, it often makes sense to first test the whole model development process and then add the tests for the deployment. This is especially important since both parts target different environments as described in section 1.2. [5]

The load tests are testing the robustness of the application by simulating a real-world load. This measures the following factors: [26]

- Endurance: Test if the application can resist the expected load for an extended period.
- Volume: Test if the application can handle a large volume.
- Performance: Test if the application performs stable with a certain workload.
- Scalability: Test if the application is able to scale up or down depending on the number of requests.

This can be of high importance, since the size of the model can vary greatly depending on the used parameters, e.g., number of trees and tree depth for tree-based models. Finally, the full application should be tested manually from end to end which requires manual effort and thus comes with the highest costs. [5]

All of the described tests are relevant for the MLOps approach. It is important to note that not all of them are executed exclusively in the integration environment which focuses on testing of the code. The tests regarding data quality, model performance or load testing are also essential in the prod environment. This is important since the lifecycle of code, data and model are not the same, e.g., one could train a new model although the code has not changed and thus the tests in the integration environment were not executed. How the tests are integrated into the MLOps process is described in the next chapter.

2 The MLOps Process

In this chapter the theory of the model development and deployment steps of the MLOps process are described in detail. To understand a complete MLOps Process one must first define a typical ML process. The key features of a such a process are categorized in Figure 4 and will be explained in the following in detail.

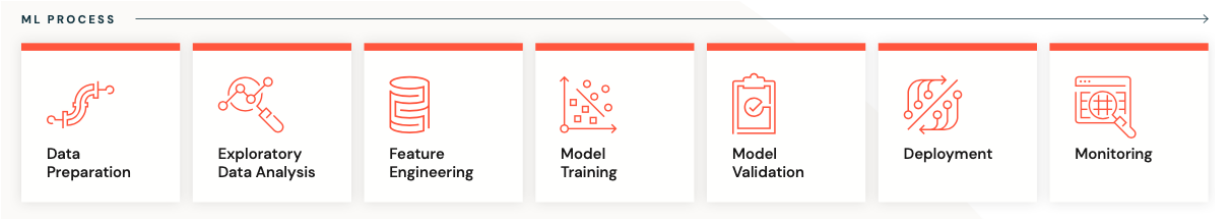

Figure 4: ML process (reprinted from [13])

The question at hand is how to bring this process together with the principles of DevOps and a valid testing strategy to derive an automated workflow out of it. The proposed framework is based on MLOps processes which already consider the usage of different environments to some extent. [13], [22] However, these frameworks are now extended and adapted to allocate the required tests to specific steps and environments of the process. Especially the complexity of different tests and staging environments for the model development and serving process was not addressed in the existing literature. This is even increased since the process must account for the different lifecycles of model, data, and code.

The complete workflow is shown in Figure 5. Here the three environments for the model development process are shown. First the code to develop a model is created in the development environment. Then the CI process starts in the integration environment before the code is ready to be released. Once a new model development process is triggered a latest released code is used to develop a new which is finally stored in the model registry in the production environment. This model is then tested together with the application code in the integration serving environment and finally deployed in the production serving environment (CD). The application is constantly monitored through the produced logs and stores the input and output data which can then be used for a new development.

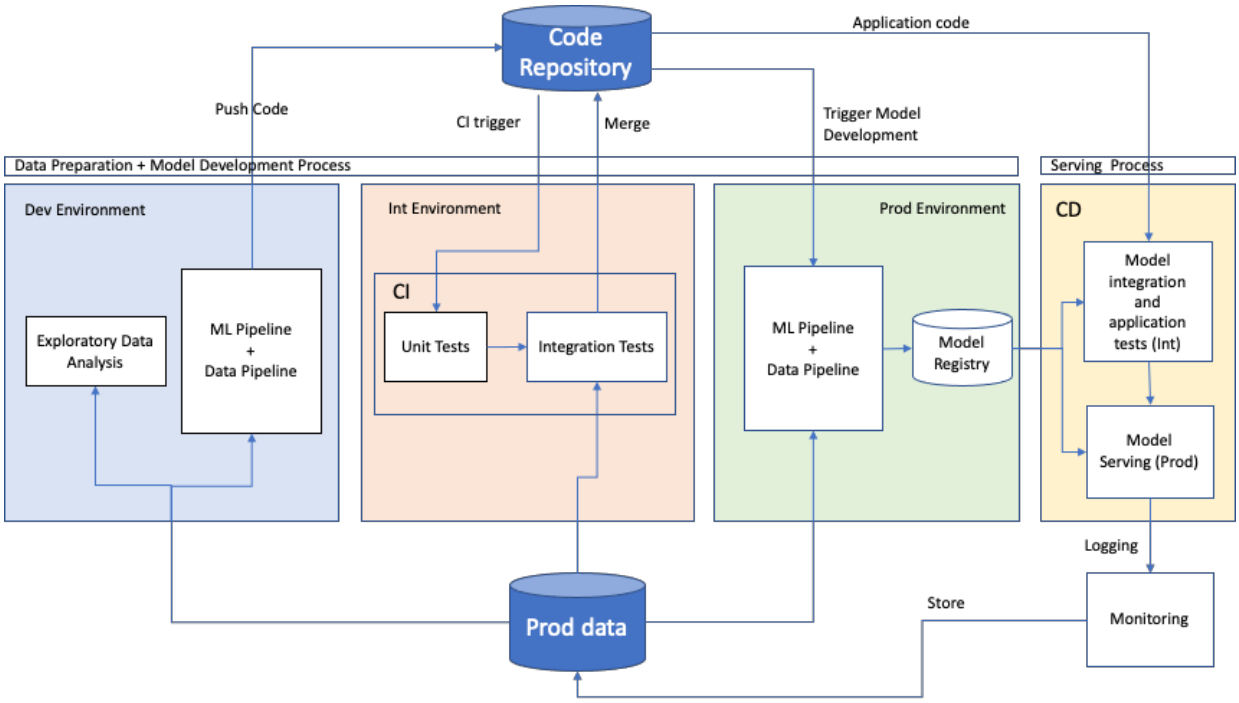

Figure 5: Proposed MLOps Process (adapted from [13])

As Figure 5 only provides a high-level overview over the different steps taken in each environment, these will now be described in more detail. Special focus is paid to the following requirements to the MLOps process:

- Reproducibility: In order to being able to reproduce a model it is necessary to control the version of all components and parameters used during the model development. This includes the data, the code, the model, and the software environment. [22]
- Reliability: Automatization of the whole workflow is only possible if it can be ensured that all processes and tasks function as expected and fit together. This means extensive testing of the project which is not limited to the CI process as in the DevOps framework.
- Lifecycle of model, code, and data: Since these three components are the main part of the MLOps workflow it is important to understand, that their lifecycles often operate asynchronously. For example, one might want to trigger the development of a new model before pushing changes to the code and vice versa. This increases the need for similar tests at different points of the process. [13]

2.1 Development Environment - Model Development

In the development environment most of the data scientists work happens, i.e., the code used for the ML process is written here. Hence, in case code or configurations regarding the ML model need to be changed this will be done in this environment. Since the model will heavily

depend on the used data and the Data Scientist cannot know beforehand which data will be useful real production data must be available. Other options, like working on artificial data, are not applicable. Since this is not a typical requirement to development environments the IT infrastructure needs to take special care to fulfil it.
The Figure 6 illustrates the process in the development environment and will be discussed in the following in detail.

The first step here is to get a general understanding of the data. Since the model's performance greatly depends on the data used, the Data Scientist has to **explore and analyze the data**. For this often the help of a business expert is required to answer specific questions which come up during this process.
To avoid unnecessary work by training models on flawed data or preparing data based on wrong assumptions, it is important to validate the quality of the data as early as possible. As described in section 1.3 the data tests are defined here. Also, a schema can be derived from the available data with the help of a domain expert to be used for further tests of new and unseen data.

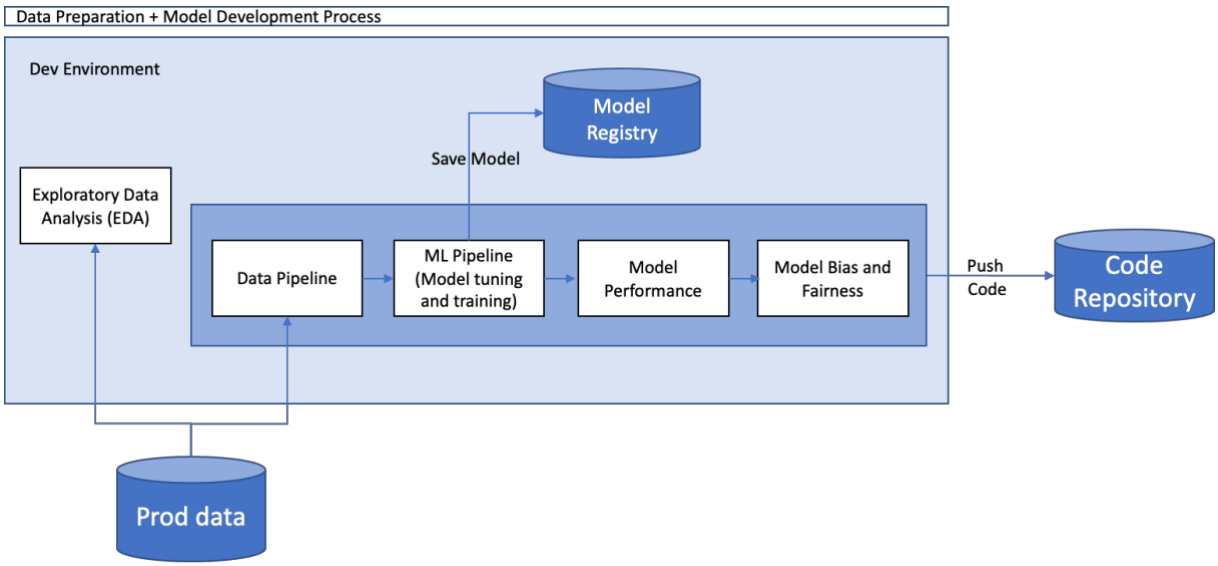

Figure 6: Model development process on Dev (adapted from [13])

Furthermore, newly developed features are analyzed. New feature can be derived through: [29]

- Extraction: Derive new information from the existing information.
- Enrichment: Include new external information.
- Combination: Combine different features.
- Encoding: Describe the feature in a different way.

It is important to understand that there is a significant difference between the exploratory data analysis (EDA) and the following processes, namely data preparation and model training. This first experimentation step will not be deployed to the integration and production environment. Thus, it does not have to fulfill the requirements imposed by these environments. The main goal of the EDA is to fasten the development process and gain insights which can be translated into the production code. In this experimental step it is also possible to conduct a feasibility study if the goal of the project is achievable with the data. In contrast developing production code for data preparation and model training requires much more effort. Production ready code in data science projects should fulfil the following requirements: [12], [34]

- Reproducible: Code should be versioned with a reproducible environment.
- Documented: Code and structure should follow a general convention and the usage and steps should be documented so that is possible for other people to understand it.
- Modular: Code should be written in a robust way, easily executable and transferrable to different environments.
- Reliable: Usage of unit and integration tests to ensure high quality of code as described in section 1.3.

During the **data pipeline** the rules are defined to transform and clean the data to bring it into a usable format. Here one uses the insights gained from the EDA and brings it into a structured and modularized format.
During the first training of the model these rules are iteratively reworked as feedback from the model training and tuning comes in. In the production environment when the process of training a new model is automated then the defined rules are also used. However, if there is evidence that the defined rules do not work in production e.g., through monitoring, then the rules must be adjusted manually.
The **ML pipeline** uses the features from the previous step to train one or several models. First the data is split into three components, namely training, validation and test. Then the hyperparameter tuning is performed by training the models with different combinations of parameters and evaluating the results based on the validation sample. With this the best performing algorithm and set of hyperparameters is determined.
In case the model training is not very computationally intensive it also makes sense to combine the training and validation data and use cross validation for evaluating the model to prevent stochastic errors in the validation. However, for computational costly trainings one has to balance the number of parameter combinations to test against the potential error coming from not using cross validation.
At this point it is important to keep in mind that this evaluation could also include different technical aspects. For example, an increase in the model complexity will lead to a bigger size

of the model. This will have a direct effect on the functionality in production in terms of inference speed, need of computational power and thus energy consumption. Also, in case different features are used and thus the model input changes (e.g., with a feature selection method as part of the development process) there will be additional efforts bringing the model to production as the input schema needs to be adjusted. This tradeoff between complexity and benefit must be evaluated individually based on the business context.

At the end the selected model is trained on the combination of training and validation sample and validated on the test sample. This gives the final estimate of the performance of the model. This is used to check if the model satisfies the required quality. These insights in the development environment can also be used to define certain benchmarks for further evaluations of the model performance. Then further evaluations on model bias and fairness are performed as already discussed in section 1.3.

Considering the evaluation and comparison of models it is key to ensure the administration and **reproducibility** of different model versions. This is on the one hand important for data scientists to be able to return to a certain state in case the experiments led to a lower quality. On the other hand, models and decisions must be made transparent to audit-teams even long time after development. This is accomplished with the help of the model registry. In order to keep track of all the trained models and thus of the decisions taken to select a certain model the model registry stores the ML models as artifacts together with their metadata. Later, this information can be used to verify why specific decisions were taken and thus solving the problem of auditability as well as using the metainformation to reproduce a certain model. Furthermore, for a successful retraining of a model the used data and code must have version control with the respective versions being stored as metainformation for each model. For code versioning tools like GitHub are well established and can be used. For data different approaches exist. For example, the data can just be stored as artifacts together with the model or special data versioning tools like dvc (data version control) [3], [36], pachyderm [23] or delta tables [1] can be used.

The need for reproducibility already applies to the development environment to justify certain decisions, however versioning of data might not be allowed for security reasons since production data is used inside a development environment[2]. Reproducibility is especially critical during the training of the final model in the production environment. Here the possibility of retraining a model is important to mitigate the risk of a possible deletion of the model in production and ensuring auditability.

[2] See section 1.2 for details about the difference in the environments.

Once the code changes are implemented and the appropriate tests are created the complete code is pushed to a Code Repository, like GitHub. This ensures that the code is versioned and stored in a central place.

2.2 Integration Environment – Preparation for Production

The main purpose of the integration environment is to ensure that the developed code is of high quality and the different parts of the pipeline function together, e.g., the prepared data can be fed into the ML pipeline. Here the tests are executed in a separate environment as key part of Continuous Integration.

The heart of this subprocess are unit and integration tests. It will now be discussed how the tests described in section 1.3 are carried out. It is important to remember that the focus lies in testing the code used for transforming the data and developing the model and not to derive a model of high performance.

Figure 7 shows the process of executing the required tests. Usually, each push or pull request (PR) in the code repository will trigger a new run of the CI pipeline. This will check out the respective development and test code and deploy and run it in the integration environment.

The first tests executed are the unit tests of each functionality. As these tests should be independent of the environment it is recommended to execute them as early as possible. Since these tests come usually with low costs and are independent of the execution environment, they can be executed with every change of code, i.e., directly after a push to a Code Repository.

Furthermore, the complete model development process should be integration tested from end-to-end. This is executed once the unit tests have passed and the code is requested to be merged. This includes the complete data pipeline and ML pipeline. The functionality of the model performance as well as the model bias and fairness are checked. The executed tests also include testing if the model input and output schema are valid, and the model logs the required information as explained in section 1.3.

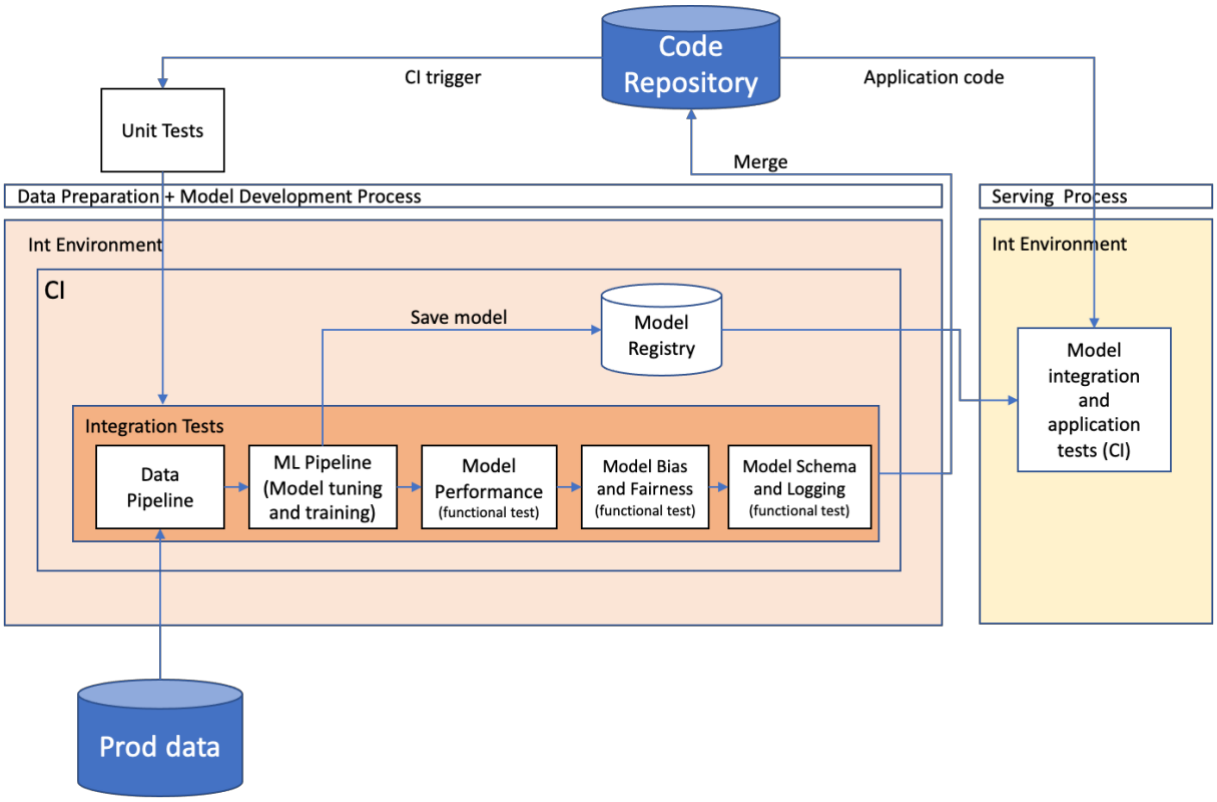

Figure 7: MLOps testing workflow on Int (adapted from [13])

If possible, in terms of costs and team structure the integration of the model with the application is tested here as well. This means deploying the developed model together with the application code to a separate test environment which mimics the real production environment of the application.

Since the main purpose of this process is testing of code and functionalities together with the correct data schemas it is not important to train a valid model but rather focus on functionality and speed. Hence, testing of model performance and data quality is not part of this process but only the functionality behind it. This means it often makes sense to either use artificial data or only a subset of the real data to accelerate the testing process and fulfil security standards. Also using fewer iterations for training could accelerate the process and thus save time and costs.

After successfully passing all the tests, the code can be merged into the main branch and is now regarded as production code.

2.3 Production Environment – Deployment

In the production environment the model development process as described in the development environment (section 2.1) is executed. As a result, a final model artifact is produced and stored in the model registry.

It is essential to understand that it should be possible to trigger the development of a new model at any time, independent of new features which are developed in parallel. This is why it is important, that all code executed here is already fully tested and additionally further tests are executed. The complete process can be viewed in Figure 8.

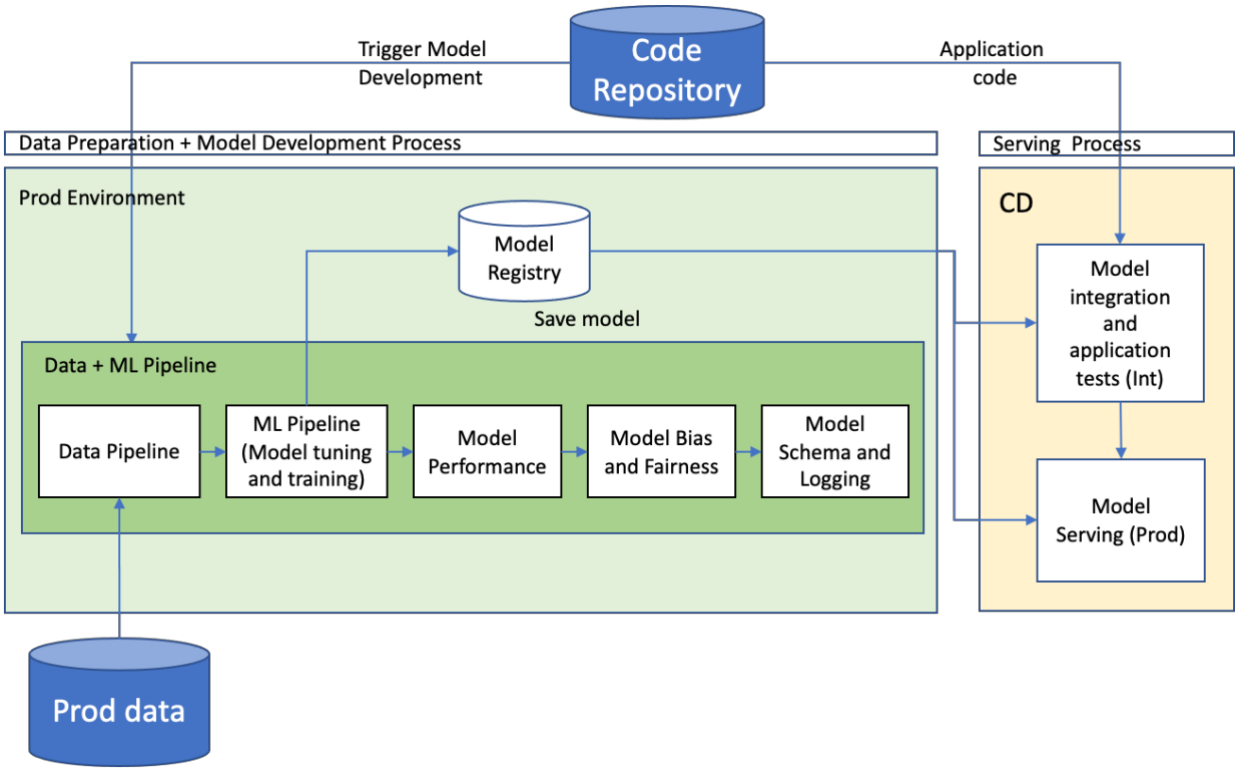

Figure 8: Model development and deployment on Prod (adapted from [13])

In contrast to the integration environment where code and functionalities are tested in this process the qualitative tests are executed. This means that the data quality checks ensure that suitable data is used for the development and the model performance tests confirm the quality, fairness, and correct formatting of the model amongst others as described in section 1.3. Here the information and knowledge regarding schemas and benchmarks from the development environment are used.

After the model has been trained and evaluated it is wrapped as an ML artifact which defines the handover object to the deployment pipeline. This artefact contains: [22]

- The code used to develop the model or a direct link to the specific version and location.
- The data used to train and validate the model or a direct link to the source path with respective version.
- The configuration of the model parameters.
- The final model in an executable format.
- The environment specification including the used libraries with their specific version.

- Required Documentation.

This information is required for being able to use the model within the application. Now the continuous delivery pipeline starts. First, the load tests with the final model in combination with the application are tested to ensure technical functionality similar to the tests executed in integration environment. Here it becomes apparent why the integration environment of the serving process needs to mimic the production environment. In case these would be different the testing of the infrastructure with load tests would be meaningless. If these are carried out successfully the model can be released.

There are two basic scenarios how models are served and consumed in production: [22], [27]

- Batch inference: Here the model is used to determine the predictions for a complete dataset. For large datasets different parallelization techniques like Apache Spark can be used to increase the computation speed. The batch scoring is usually performed for scheduled jobs.
- Real-Time inference: Here the model is used to determine the prediction of a single or small observations in real time. In this case multiple instances of the model can be deployed, and this requires parallelization by a load balancer.

There are different ways a smooth update of the existing model in production can happen. This is especially important in case of real-time inference as here restarting of the server should be avoided to reduce downtimes. Often, the new version of the model is deployed in parallel to the existing, stable one. Once the functioning of the new model is ensured the workload is shifted and the old version is shut down. In reality there are a lot of different modifications of this framework like canary deployments or blue-green deployments. [22]

In additional to the way how a model is used in production it is also import how the model is integrated into the application itself. There are three different patterns how this can be achieved: [11], [27]

- Embedded model: Here the model is included in the application code as normal dependency. This means the application code together with the trained model can be treated as one application artifact and version. This pattern requires the model to be in a readable format by the application. This can be challenging in case different programming languages are used. For example, one can use the framework H2O to export the model as a Java object. [32] In case of a python model the package ‘pex’ (python executable) can be used to transform the model into an executable which can then be embedded into the application. [24]
- Model deployed as a standalone service: The model is deployed independent of the consuming application. However, the standalone service will still need some additional

code for pre- and post-processing of the data. This pattern is often supported by many cloud providers. They offer different tools to simplify the deployment of the model in their Machine Learning as a Service Platforms like Azure Machine Learning or AWS Sagemaker.

- Model deployed at runtime as data: Here the model is also deployed independently but the consuming application will load it at runtime as data. This is especially useful in streaming applications. These are able to load the new model into memory while still using the previous version.

In contrast to the 'model-as-a-service' pattern the 'embedded-model' and 'model-as-data' pattern have a high impact on the dependencies of the model which emphasizes the importance of testing the model integration.
This means the correct usage of a model in production requires additional information besides the model itself. It also requires saving the information about the environment, like version of the programming language and dependent libraries with their respective versions.
During the deployment it is important to replicate this environment. As already discussed, an additional requirement comes from the fact that in the serving environment multiple models with incompatible dependencies can be deployed. Furthermore, different models can compete for the available resources which means a new model will have direct effects also on other models. In case the model integration is not tested beforehand and not monitored appropriately, this can lead to significant declines in execution speed and even lead to an unavailability of the service itself.
These challenges are usually addressed by containerization. Here components of an application like libraries or configurations are bundled into a single container image. This container image can be run in different operating systems. [22]
The most prominent technology used in this context is the open-source framework Docker. [33] In combination with Kubernetes, which is able to orchestrate different containers efficiently, it provides a powerful infrastructure for the hosting of applications. [17]
The proposed processes solve the problem of asynchronous lifecycles of data, model and code by testing the code component for the model development as suggested by DevOps in an integration environment. The quality of the data and model are then additionally tested during the development in the prod environment. Special attention and care must be paid to the testing of the model integration into the application which is part of the CD process.

2.4 Monitoring

To ensure the continuous functioning of an ML model in production there are two main drivers which need to be monitored: [22]

- The consumed resources: This monitors if the infrastructure is working fine. It means one has to constantly check if the used processors, memory etc. behave as expected. Especially in times of high demands it should be evaluated if the system can scale the used resources accordingly to keep the execution time in the required boundaries.
- The performance of the model: Here it is evaluated if the performance metrics of the model deviate from the development or deteriorate over time. In this case finding the cause can be complicated as the performance strongly depends on the incoming data. One explanation for lower model performance can be a systematic drift of the data. Hence, the distribution of the data needs to be monitored as well.

One key component to be able to get the required information for a successful monitoring is an effective logging-system which is even addressed by legislation. [9] To fulfil the requirements the application should at least provide the following information for each run: [22]

- Model-meta-data: Information to identify which model was used.
- Model-input: Value of the features used to call the application.
- Model-output: Prediction of the model.
- Response of the application: In case the application triggers a specific measurement based on the model-output this measure needs to be provided.
- Explainability of the model: In case required by regulators specific information about the explanation of the prediction should be logged.

The collected information should be analyzed to extract different measures. Such measures could include shutting down old versions of the model, e.g., in the case of parallel deployments. Another measure would be using observations from production to retrain and improve the model in case of performance deterioration. This feedback loop is of great importance to effectively complete the whole process since it can automatically trigger a new deployment. [11]

3 Conclusion

The aim of this paper was to explain the MLOps process with its underlying theory and propose a general framework. In detail a summary of the necessary tests to ensure a reliable process were described. The proposed framework extends the existing literature by integrating these tests in specific steps of the process. Especially the complexity of combining different tests and environments together with asynchronous lifecycles of data, model and code was addressed. The framework MLOps emerged from DevOps which is a software engineering practice focusing on CI/CD. However, due to additional complexity by using data and model as

additional components to code further adaptions are needed. Next, the concept of execution environments was introduced to emphasis the need for different environments for developing, testing and execution of code. Usually, there are separated environments used both, for the development of the model and for the deployment of the application. The described test strategy covers all aspects of data, code and model relevant for the development of the model and its deployment. A test pyramid was introduced which showed the low-cost unit tests for the code, the qualitative tests for model and data as well as the integration tests of the whole pipeline.

The proposed MLOps framework is using these concepts to fulfil the requirements of reproducibility, reliability and life cycle management. First, the data is analyzed, and the insights are integrated into the model development code. This code should later be able to run at any time in the production environment an create new models on demand. Thus, a suitable testing of the code and the whole process to guarantee its functionality is required in the integration environment. Special attention must be provided to the integration of the model into the application. In the production environment the new model is trained leveraging the knowledge gained in the development environment. Different tests regarding data and model quality are executed. Finally, the application with the new model is tested and then deployed. Here different deployment strategies can be applied. To ensure a continuous functioning of the application an appropriate monitoring is required.

Bibliography

[1] Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., ... & Zaharia, M. (2020). Delta lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment, 13*(12), 3411-3424.

[2] Atlassian (2023). *History of DevOps*. URL: https://www.atlassian.com/devops/what-is-devops/history-of-devops (accessed 23.02.2023).

[3] Banerjee, I., Ghanta, D., Nautiyal, G., Sanchana, P., Katageri, P., & Modi, A. (2023). MLOps with enhanced performance control and observability. *arXiv preprint arXiv:2302.01061*.

[4] Baumgartner, M., Klonk, M., Pichler, H., Seidl, R., & Tanczos, S. (2021). *Agile Testing*. Springer International Publishing.

[5] Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017, December). The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 1123-1132). IEEE.

[6] Chattopadhyay, S., Prasad, I., Henley, A. Z., Sarma, A., & Barik, T. (2020, April). What's wrong with computational notebooks? Pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI conference on human factors in computing systems* (pp. 1-12).

[7] Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *Ieee Software*, *33*(3), 94-100.

[8] Erich, F., Amrit, C., & Daneva, M. (2014). Report: Devops literature review. *University of Twente, Tech. Rep*.

[9] European Commission. (2021). Laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain union legislative acts.

[10] Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2021, December). On continuous integration/continuous delivery for automated deployment of machine learning models using mlops. In *2021 IEEE fourth international conference on artificial intelligence and knowledge engineering (AIKE)* (pp. 25-28). IEEE.

[11] Granlund, T., Stirbu, V., & Mikkonen, T. (2021). Towards regulatory-compliant MLOps: Oravizio's journey from a machine learning experiment to a deployed certified medical product. *SN computer Science*, *2*(5), 342.

[12] Huijskens, T. (2019). *Data Scientists, the Only Useful Code is Production Code*. URL: https://medium.com/quantumblack/data-scientists-the-only-useful-code-is-production-code-8b4806f2fe75 (accessed 25.04.2023).

[13] Joseph, B., Rafi, K., Matt, T., Niall, T. (2022). *The Big Book of MLOps*. Databricks.

[14] JWT (2023). *Introduction to Json Web Tokens*. URL: https://jwt.io/introduction (accessed 22.04.2023).

[15] Karamitsos, I., Albarhami, S., & Apostolopoulos, C. (2020). Applying DevOps practices of continuous automation for machine learning. *Information*, *11*(7), 363.

[16] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). *Machine learning operations (mlops): Overview, definition, and architecture.* IEEE Access.

[17] Kubernetes, T. (2019). Kubernetes. *Kubernetes.*

[18] Liu, Y., Zaharia, M. (2022). *Practical Deep learning at scale with mlflow*. Packt Publishing.

[19] Masse, M. (2011). *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.".

[20] Mercedes Benz Mobility. *About Us.* URL: https://www.mercedes-benz-mobility.com/en/who-we-are/about-us/ (accessed 11.02.2023).

[21] Murphy, C., Kaiser, G. E., & Arias, M. (2007). *An approach to software testing of machine learning applications.*

[22] Omont, N., Treveil, M., Stenac, C., Lefèvre, K. (2021). *MLOps – Kernkonzepte im Überblick: Machine-Learning-Prozesse im Unternehmen nachhaltig automatisieren und skalieren*. Germany: O'Reilly.

[23] Pachyderm (2023). *Getting-Started*. URL: https://docs.pachyderm.com/latest/getting-started/ (accessed 27.04.2023).

[24] Pex (2023). Building .pex *files*. URL: https://pex.readthedocs.io/en/v2.1.134/buildingpex.html (accessed 26.04.2023).

[25] Perez, F., Granger, B. E., & Hunter, J. D. (2010). Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, *13*(2), 13-21.

[26] Raj E. (2021). *MLOps-Entwicklung: Schnelles und skalierbares Entwickeln, Testen und Verwalten produktionsbereiter Machine-Learning-Lebenszyklen*. Packt Publishing.

[27] Sato, D, Wilder, A, Windheuser, C. (2019). C. *Continuous delivery for machine learning*. URL: https://martinfowler.com/articles/cd4ml.html (accessed 20.04.2023).

[28] Scikit-learn (2023). *Getting Started*. Url: https://scikit-learn.org/stable/getting_ started.html (accessed 20.04.2023).

[29] Seiter, M. (2019). *Business Analytics*. Vahlen.

[30] Subramanya, R., Sierla, S., & Vyatkin, V. (2022). From DevOps to MLOps: Overview and Application to Electricity Market Forecasting. *Applied Sciences*, *12*(19), 9851.

[31] Tiangolo (2023). *FastApi*. URL: https://fastapi.tiangolo.com (accessed 15.04.2023).

[32] Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., & Farivar, R. (2019, November). Towards automated machine learning: Evaluation and comparison of

AutoML approaches and tools. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)* (pp. 1471-1479). IEEE.

[33] Turnbull, J. (2014). *The Docker Book: Containerization is the new virtualization*. James Turnbull.

[34] Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., ... & Wilson, P. (2014). Best practices for scientific computing. *PLoS biology*, *12*(1), e1001745.

[35] Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39-45.

[36] Zhao, Y., Belloum, A. S., & Zhao, Z. (2022). MLOps Scaling Machine Learning Lifecycle in an Industrial Setting. *International Journal of Industrial and Manufacturing Engineering*, *16*(5), 143-153.