applied sciences

MDPI

Article

Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools

Philipp Ruf [1,†], Manav Madan [1,†], Christoph Reich [1,*] and Djaffar Ould-Abdeslam [2]

[1] Institute for Data Science, Cloud Computing and IT-Security (IDACUS), Hochschule Furtwangen University, 78120 Furtwangen im Schwarzwald, Germany; philipp.ruf@hs-furtwangen.de (P.R.); manav.madan@hs-furtwangen.de (M.M.)
[2] Institut de Recherche en Informatique, Mathématiques, Automatique et Signal (IRIMAS), Université de Haute-Alsace, 61 Rue Albert Camus, 68093 Mulhouse, France; djaffar.ould-abdeslam@uha.fr
* Correspondence: christoph.reich@hs-furtwangen.de
† These authors contributed equally to this work.

Abstract: Nowadays, machine learning projects have become more and more relevant to various real-world use cases. The success of complex Neural Network models depends upon many factors, as the requirement for structured and machine learning-centric project development management arises. Due to the multitude of tools available for different operational phases, responsibilities and requirements become more and more unclear. In this work, Machine Learning Operations (MLOps) technologies and tools for every part of the overall project pipeline, as well as involved roles, are examined and clearly defined. With the focus on the inter-connectivity of specific tools and comparison by well-selected requirements of MLOps, model performance, input data, and system quality metrics are briefly discussed. By identifying aspects of machine learning, which can be reused from project to project, open-source tools which help in specific parts of the pipeline, and possible combinations, an overview of support in MLOps is given. Deep learning has revolutionized the field of Image processing, and building an automated machine learning workflow for object detection is of great interest for many organizations. For this, a simple MLOps workflow for object detection with images is portrayed.

Keywords: MlOps; tool comparison; workflow automation; quality metrics

Citation: Ruf, P.; Madan, M.; Reich, C., Ould-Abdeslam, D. Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools. *Appl. Sci.* **2021**, *11*, 8861. https://doi.org/10.3390/app11198861

Academic Editor: Francesco Bianconi

Received: 2 September 2021
Accepted: 18 September 2021
Published: 23 September 2021

1. Introduction

Across multiple industries, new applications and products are becoming increasingly Machine Learning (ML)-centric. In general, ML is still a tiny part of a larger ecosystem of technologies involved in an ML-based software system. ML systems, i.e., a software system incorporating ML as one of its parts, are known to add several new aspects previously not known in the software development landscape. Furthermore, manual ML workflows are a significant source of high technical debt [1]. Unlike standard software, ML systems have complex entanglement with the data on top of standard code. This complex relationship makes such systems much harder to maintain in the long run. Furthermore, different experts (application developers, data scientists, domain engineers, etc.) have to work together. They have different temporal development cycles and tooling, and data management is becoming the new focus in ML-based systems. As a reaction to these challenges, the area of Machine Learning Operations (MLOps) emerged. MLOps can help reduce this technical debt as it promotes automation of all the steps involved in the construction of an ML system from development to deployment. Formally, MLOps is a discipline which is formed of a combination of ML, Development Operations (DevOps), and Data Engineering to deploy ML systems reliably and efficiently [2,3].

Advances in deep learning have promoted the broad adoption of AI-based image analysis systems. Some systems specialize in utilizing images of an object to quantify the

quality while capturing defects such as scratches and broken edges. Building an MLOps workflow for object detection is a laborious task. The challenges one encounters when adopting MLOps are directly linked with the complexity due to the high dimensionality of the data involved in the process.

Data are a core part of any ML system, and generally, they can be divided into three categories, i.e., structured, semi-structured, and unstructured data. Vision data, i.e., images and videos, are considered as unstructured data [4]. Analyzing images and videos by deep neural networks (DNNs) like FastMask has gained much popularity in the recent past [5]. Nevertheless, processing images with ML is a resource-intensive and complex process. As the training of DNNs requires a considerable amount of data, automated data quality assessment is a critical step in an MLOps workflow. It is known that if the quality of data is compromised to some specific level, this makes a system more prone to failure [6]. For images, it is challenging to define metrics for automated quality assessment because metrics such as completeness, consistency, etc., cannot be universally defined for image data sets. On the other hand, structured data are easier to process as one can clearly define data types, quality ratings (set by domain experts), and quality assessment tests.

Moreover, several tools are available for building an automated MLOps workflow. These tools can be used to achieve the best outcomes from developing of a model to deployment until maintenance. In most cases creating an MLOps workflow requires multiple tools which collaborate to fulfill individual parts.

There is also a high overlap of functionalities provided by many of these tools. The selection of tools for an optimal MLOps pipeline is a tedious process. The recipe for the selection of these tools is the requirements that each step of the workflow introduces. This recipe is also dependent on the maturity level of the machine learning workflow adopted in an organization and the capabilities of integration of each tool or ingredient. Therefore, we address these challenges in the work on hand. The main contributions of this paper are as follows:

- A holistic analysis and depiction of the need for principles of MLOps.
- An intensive consideration of related roles in MLOps workflows and their responsibilities.
- A comprehensive comparison of different supportive open-source MLOps tools, to allow organizations to make an informed decision.
- An example workflow of object detection with deep learning that shows how a simple GitFlow-based software development process can be extended for MLOps with selected MLOps tools.

Demystifying MLOps and selection procedure of tools is of utter importance as the complexity of ML systems grows with time. Clarification of stages, roles, and tools is required so that everyone involved in the workflow can understand their part in the development of whole system.

This paper is structured as follows. In Section 2, a quick overview of work mentioning and using MLOps techniques is given and is followed by the depiction of workflows in MLOps in Section 3. With a description of the various roles in Section 4 and a comparison of MLOps supporting tools and applicable monitoring metrics in Section 5, a use case for automating the workflow for object detection with deep neural nets is discussed in Section 6. With potential future research directions, this work is concluded in Section 7.

2. Related Work and State-Of-The-Art

As MLOps is a relatively new phenomenon, there is not enough related work, implementation guidance, specific instructions for starting such project structures, or experience reports. Therefore, we outline work on different aspects of MLOps in the following, paying particular attention to holistic MLOps workflows, data quality metrics, and ML automation. By addressing these core aspects during the work on hand, we aim to clarify the potentials and practical applications of MLOps.

Trends and challenges of MLOps were summarized by Tamburri in [7]. In this paper, MLOps is defined as the distribution of a set of software components realizing five ML pipeline functions: data ingestion, data transformation, continuous ML model (re-)training, (re-)deployment, and output presentation. By further discussing trends in artificial intelligence software operations such as explanation or accountability, an overview of state-of-the-art in MLOps is given. While the author defined and discussed trends and challenges in sustainable MLOps, the work at hand also covers a comparative overview of responsibilities and possible tool combinations concerning image processing. A framework for scalable fleet-analytic (e.g., ML in distributed systems) was proposed by Raj et al. in [3], facilitating ML-Ops techniques with focus on edge devices in Internet of Things (IoT). The proposed scalable architecture was utilized and evaluated concerning efficiency and robustness in experiments covering air quality prediction in various campus rooms by applying the models on the dedicated edge devices. As the formulated distributed scenario underlines the demand for a well-defined MLOps workflow, the authors briefly introduced the applied modeling approach. Next to the focus on IoT devices, the work on hand differs due to an extensive discussion of applicable metrics, utilization of tools while experimenting, and a definition of involved actors and their responsibilities. Fursin et al. proposed an open platform *CodeReef* for mobile MLOps in [8]. The non-intrusive and complementary concept allows the interconnection of many established ML tools for creating specific, reproducible, and portable ML workflows. The authors aimed to share various parts of an MLOps project and benchmark the created workflow within one platform. In contrast, this work considers multiple applicable solutions to the various steps in such workflows. Two real-world multi-organization MLOps setups were presented by Granlud et al. in [9]. With a focus on use cases that require splitting the MLOps process between the involved organizations, the integration between two organizations and the scaling of ML to multi-organizational context were addressed. The physical distribution of ML model training and the deployment of resulting models are not explicitly responded to by the work on hand. However, one can assemble its environment boundaries, dependent on the selected toolchain. Although both scenarios stated by the authors were highly domain-specific, the elementary formulation and differentiation of ML and deployment pipelines overlap with this work. Zhao et al. reviewed the literature of MLOps. Briefly, they identified various challenges of MLOps in ML projects, differences to DevOps, and how to apply MLOps in production in [10]. Muralidhar et al. summarized commonly practiced antipatterns, solutions, and future directions of MLOps in [11] for financial analytics. In applying the stated recommendations, error sources in ML projects can be reduced. Next to conforming with the stated collateral best practices in MLOps workflows, the work on hand also tries to extend and generalize the author's definition of involved actors, specific to the financial use case.

An approach for easy-to-use monitoring of ML pipeline tasks, while also considering hardware resources, was presented by Silva et al. in [12]. Concerning the required time for completing tasks (e.g., operations in batch or stream processing data sets) and focus on resource utilization, thoughts on benchmarking ML-based solutions in production were given. The authors overall compared nine datasets consistent with binary and multiclass regression problems, where three datasets were based on images. Their approach was evaluated by benchmarking all datasets in batch mode and applying five datasets for online tasks with every 50 and 350 threads. Monitoring aspects of resources in the different stages of an MLOps workflow is included in the work on hand, and there is no focus on benchmarking the chosen toolchain environment. Concerning data warehousing applications, Sureddy and Yallamula proposed a monitoring framework in [13] . In defining various requirements for monitoring such complex and distributed systems, the framework helps in building effective monitoring tools. As MLOps systems require a monitoring solution, too, aspects of monitoring the server as well as the application are treated in the work on hand. Various aspects and best practices of monitoring the performance and planning infrastructure for specific projects were outlined by Shivakumar in [14]. In considering

the server- as well as the application side of such undertakings, a CICD sample setup and sample strategy of using commercial tools for infrastructure monitoring in a disaster recovery scenario were given. As the monitoring and CICD aspects are an inevitable part of MLOps, the work on hand is not concerned with disaster recovery. A definition of measuring data quality dimensions as well as challenges while applying monitoring tools was outlined by Coleman in [15]. By translating user-defined constraints into metrics, a framework for unit tests for data was proposed by Schelter et al. in [16]. Using a declarative Application Programming Interface (API) which consumes user-defined validation codes, the incremental- and batch-based assessment of data quality was evaluated on growing data sets. Regarding the data completeness, consistency, and statistics, the authors proposed a set of constraints and the respective computable quality metrics. Although the principles of a declarative quality check for datasets are applicable during an MLOps workflow, the enumeration of this approach is a surrogate for the definition of quality check systems and services.

Taking the big data value chain into consideration, there are similar requirements to the domain of ML quality. Various aspects of demands on quality in big data were surveyed by Taleb et al. in [17] and answered by a proposed quality management framework. While considering the stated demands on data quality during the various sections of the work on hand, no specific framework or quality management model for big data value chains, as introduced by the authors, is proposed. Barrak et al. empirically analyzed 391 open-source projects which used Data Version Control (DVC) techniques with respect to coupling of software and DVC artifacts and their complexity evolution in [18]. Their empirical study concludes that using DVC versioning tools becomes a growing practice, even though there is a maintenance overhead. As DVC is a part of the work on hand, it neither exclusively focuses on versioning details nor takes repository and DVC-specific statistics into consideration. Practical aspects of performing ML model evaluation were given by Ramasubramanian et al. in [19] concerning a real life dataset. In describing selected metrics, the authors introduced a typical model evaluation process. While the utilization of well-known datasets is out of scope for the work on hand, the principles of ML model evaluation are picked up during the various subsequent sections. A variety of concept drift detection techniques and approaches were evaluated by Mhemood et al. in [20] with respect to time series data. While the authors gave a detailed overview of adaption algorithms and challenges during the implementation, aspects of monitoring the appearance of concept drift are picked up in work on hand. The various methods, systems, and challenges of Automated Machine Learning (AutoML) were outlined in [21]. Concerning hyperparameter optimization, meta-learning, and neural architecture search, the common foundation of AutoML frameworks is described. Subsequently, established and popular frameworks for automating ML tasks were discussed. As the automation of the various parts in an ML pipeline becomes more mature, and the framework landscape for specific problems grows, the inclusion of ML-related automation in MLOps tasks becomes more attractive. The area of AutoML was surveyed by Zöller et al. in [22]. A comprehensive overview of techniques for pipeline structure creation, Combined Algorithm Selection and Hyperparameter optimization (CASH) strategies, and feature generation is discussed by the authors. As the AutoML paradigm has the potential of performance loss while training model candidates, different patterns of improving the utilization of such systems are introduced. By discussing the shortcomings and challenges of AutoML, a broad overview of this promising discipline was given by the authors. Although we refer to specific aspects of ML automation in work on hand, models' deployment and integration into the target application are treated more intensely. Concerning research data sets, Peng et al. provided international guidelines on sharing and reusing quality information in [23]. Based on the Findable, Accessible, Interoperable, and Reusable (FAIR) principles, different data lifecycle stages and quality dimensions help in systematically processing and organizing data sets, their quality information, respectively. While determining and monitoring the quality of datasets and processed data structures are vital to different oper-

ations in MLOps, the work on hand does not address the sharing of preprocessed records. A systematic approach for utilizing MLOps principles and techniques was proposed by Raj in [24]. With a focus on monitoring ML model quality, end-to-end traceability, and continuous integration and delivery, a holistic overview of tasks in MLOps projects is given, and real-world projects are introduced. As the authors focused on practical tips for managing and implementing MLOps projects using the technologies Azure in combination with MLflow, the work on hand considers a broader selection of supportive frameworks and tools. Considering automation in MLOps, various roles and actors' main tasks are supported by interconnected tooling for the dedicated phases. Wang et al. surveyed the degree of automation required by various actors (e.g., 239 employees of an international organization) in defining a human-centric AutoML framework in [25]. With visualizing the survey answers, an overview of the different actor's thoughts on automating the various phases of end-to-end ML life cycles was given and underlined the author's assumption of only partly automating processes in such complex and error-prone projects.

Additionally, the landscape of MLOps tools has evolved massively in the last few years. There has been an emergence of high-quality software solutions in terms of both open-source and commercial options. The commercial platforms and tools available in the MLOps landscape make ML systems development more manageable. One such example is the AWS MLOps framework [26]. The framework is one of the easiest ways to get started with MLOps. The framework is built on two primary components, i.e., first, the orchestrator, and second, the AWS CodePipeline instance. It is an extendable framework that can initialize a preconfigured pipeline through a simple API call. Users are notified by email about the status of the pipeline. There are certain disadvantages of using commercial platforms for MLOps. The development process requires multiple iterations, and you might end up spending much money on a solution that is of no use. Many training runs do not produce any substantial outcome. To have a flexible and evolving workflow, it is essential to have a 100% transparent workflow, and with commercial solutions, this can not be completely ensured. In general, open-source tools are more modular and often offer higher quality than their counterparts. This is the reason why only open-source tools are benchmarked in this paper.

3. Workflow of DevOps and MLOps

With the development of hardware for the efficient development of ML systems [24], like GPUs and TPUs, software development has evolved with time. DevOps has been revolutionary for achieving this task for traditional software systems, and similarly, MLOps aims to do this for ML-centered systems. In this section, the essential terminology for DevOps and MLOps is explained.

3.1. DevOps Software Development

Software development teams have moved away from the traditional waterfall method for software development to DevOps in the recent past [24]. The waterfall method is depicted in Figure 1a. The method comprises five individual stages: Requirement Analysis, Design, Development, Testing, and Maintenance. Every individual stage of the cycle is pre-organized and executed in a non-iterative way. The process is not agile, and thus all the requirements are collected before the cycle begins, and modification in requirements is impossible. Once the requirements are implemented (Development stage), then only the testing can begin. As soon as the testing is complete, the software is deployed in production for obtaining user feedback. If, in the end, the customer is not satisfied, then the whole pipeline is repeated. Such a life cycle is not suited for dynamic projects as needed in the ML development process. To counter these disadvantages, the whole process has evolved into an agile method. Unlike the waterfall method, the agile development process is bidirectional with more feedback cycles such that the immediate changes in requirements can be incorporated faster in the product. The agile method is aimed at producing rapid

changes in code as per the need. Therefore, close collaboration between Ops and software development teams is required.

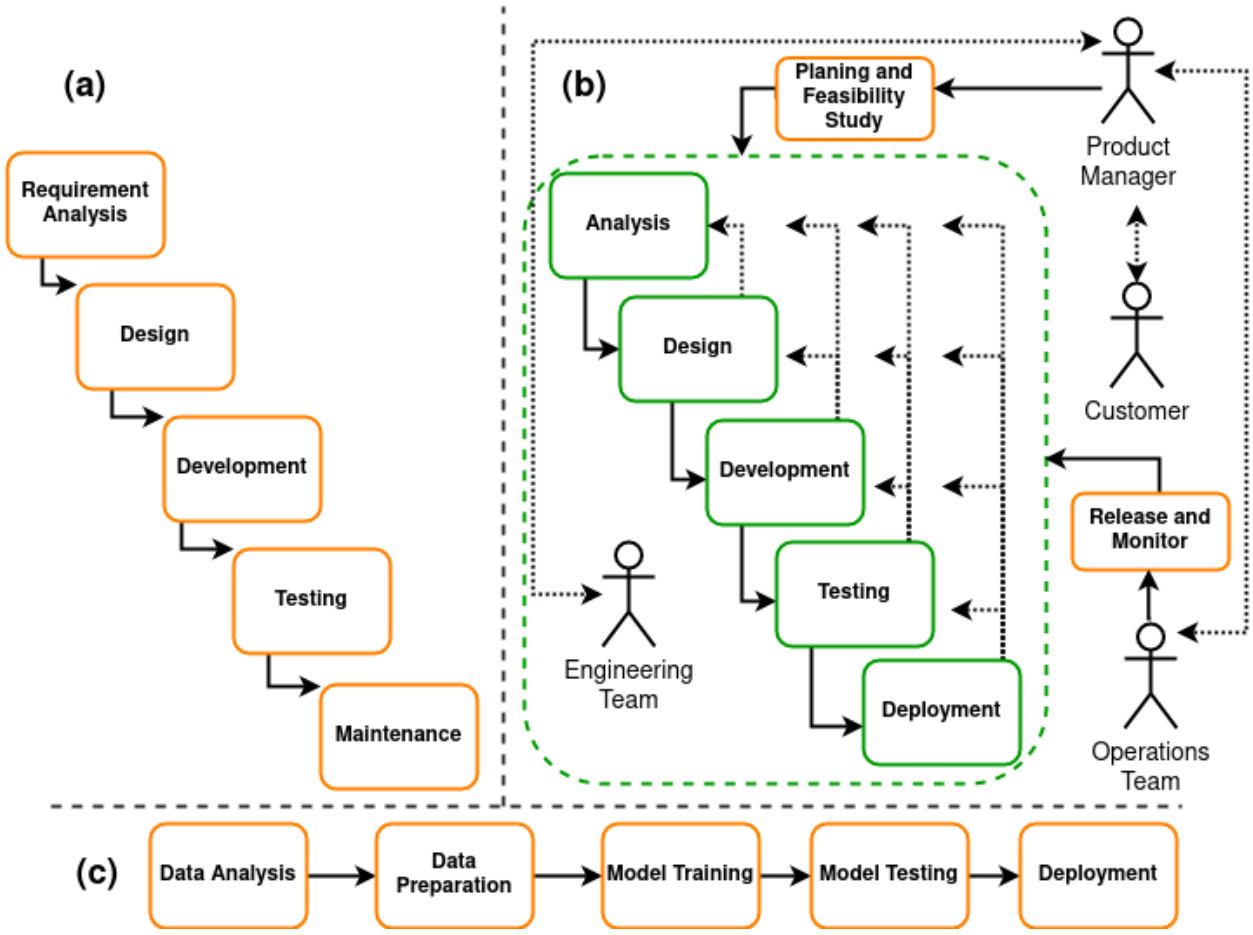

Figure 1. (**a**,**b**) Difference between Waterfall and DevOps software development life cycle. (**c**) A manual ML Pipeline.

3.2. Dev vs. Ops

After the agile methodology, DevOps emerged as the new go-to methodology for continuous software engineering. The DevOps method extended the agile development method by streamlining the flow of software change through the build, test, deploy, and delivery stages [24]. Looking at Dev and Ops individually, there was hardly any overlap between them. In the past, software development consisted of two separate functioning units, i.e., Dev (Software engineers and developers) and Ops (Operations engineers and IT specialists). The Dev was responsible for translating the business idea into code, whereas the Ops were responsible for providing a stable, fast, and responsive system to the customer at all times [27].

3.3. DevOps

Traditionally, the developers would wait until the release date to pass the newly developed code (patch) to the operations team. The operations team would then foresee that the developed code is deployed with additional infrastructure abstraction, management, and monitoring tasks. In contrast, DevOps aimed at bridging the gap between the two branches: Dev and Ops. It responds to the agile need by combining cultural philosophies, practices, and tools that focus on increasing the delivery of new features in production. It emphasizes communication, collaboration, and integration between Software Developers and Operations team [27]. An example of a DevOps workflow is depicted in Figure 1b. As seen from the Figure, the customer and project manager can redirect the development team

on short notice if there are any changes in the specification. The different phases of DevOps can be implemented in a shorter duration such that new features can be deployed rapidly. The prominent actors involved in the DevOps process are also depicted in Figure 1b.

DevOps has two core practices: Continuous Integration (CI) and Continuous Delivery (CD). Continuous Integration is a software practice that focuses on automating the process of code integration from multiple developers. In this practice, the contributors are encouraged to merge their code into the main repository more frequently. This enables shorter development cycles and improves quality, as flaws are identified very early in the process. The core of this process is a version control system and automated software building and testing process. Continuous Delivery is a practice in which the software is built in a manner that is always in a production-ready state. This ensures that changes could be released on demand quickly and safely. The goal of CD is to get the new features developed to the end user as soon as possible [27,28].

There is also another practice known as Continuous Deployment, which is often confused with CD. Continuous deployment is a practice in which every change is deployed in production automatically. However, some organizations have external approval processes for checking what should be released to the user. In such cases, Continuous delivery is considered a must, but Continuous deployment is an option that can be left out.

3.4. CI and CD Pipeline

CI and CD have been adopted as the best practices for software development in recent years. Automating these practices also requires a robust pipeline known as the DevOps pipeline or CI/CD pipeline. The pipeline consists of a series of automated processes that enable software teams to build and deploy a new version of software smoothly. It ensures that the new developments are automatically merged into the software, which is followed up by automated testing and deployment. As a fact, the DevOps with CI and CD has helped in improving and accelerating deployments of new features into production [29].

3.5. ML and ML Pipeline

An ML system is also a software system, but the development of an individual ML model is essentially experimental. A typical ML workflow starts by gathering more insights about the data and the problem at hand. The data need to be analyzed (Exploratory data analysis (EDA)), cleaned, and preprocessed (feature engineering and selection) such that essential features are extracted from the raw data (with the support of Data Stewards). This is followed by dataset formation for training, testing, and validation.

Training, validating, and testing ML algorithms is not a straightforward method. The iterative process involves fitting the hyperparameters of an ML algorithm on the training data set while using the validation set to check the performance on the data that the model has not seen before at every iteration. Finally, the test dataset is used to assess the final, unbiased performance of the model. The validation dataset is unique concerning the test dataset because it is not kept hidden from the preparation of the model, however it is instead used to give an adequate performance of the ability of the last tuned model.

At the very beginning only, these three sets of data are separated and should not be mixed at any point in time. Once the dataset is ready for training, the next step is algorithm selection and, finally, training. This step is iterative, where multiple algorithms are tried to obtain a trained model. For each algorithm, one has to optimize its hyperparameters on the training set and validate the model on the validation set, which is a time-consuming task (and implemented by a data scientist). Multiple iterations are required for acquiring the best model, and keeping track of all iterations becomes a hectic process (often, this is manually done in excel sheets). To regenerate the best results, one must use the precise configuration (hyperparameters, model architecture, dataset, etc.). Next, this trained model is tested using specific predefined criteria on the test set, and if the performance is acceptable, the model is ready for deployment. Once the model is ready, it is handed over to the operations

team, which handles the deployment and monitoring process (usually implemented by an Operation engineer) such that model inference can be done.

In the above process, all steps are incremental (similar to the waterfall software development style). These steps together form an ML pipeline as shown in Figure 1c. An ML pipeline is a script-based, manual process formed of a combination of tasks such as data analysis, data preparation, model training, model testing, and deployment. There is no automatic feedback loop and transitioning from one process to another is also manually done [29].

3.6. Operations in ML

Building an ML model is a role designated for a data scientist with the support of a Domain expert, and it does not intersect with how the business value is produced with that model. In a traditional ML development life cycle, the operations team is responsible for deploying, monitoring, and managing the model in production. Once the data scientist implements the suitable model, it is handed over to the operations team.

There are different techniques in which a trained model is deployed. The two most common techniques are "Model as a Service" and "Embedded model" [30]. In "Model as a Service", the model is exposed as Representational state transfer (REST) API endpoints, i.e., deploying the model on a web server so that it can interact through REST API and any application can obtain predictions by transferring the input data through an API call. The web server could run locally or in the cloud. On the other hand, in the "Embedded model" case, the model is packaged into an application, which is then published. This use case is practical when the model is deployed on an edge device. Note that how an ML model should be deployed is wholly based on the final user's interaction with the output generated by the model.

3.7. Challenges of Traditional ML Workflow

In comparison to conventional software development, ML has unique challenges that need to be handled differently. The major challenges encountered in the manual ML pipeline are as follows.

- ML is a metrics-driven process, and in order to select the best model, multiple rounds of experiments are performed. For each experiment, one needs to track the metrics manually which increases the overall complexity.
- Manual logging is an inefficient and error-prone process.
- Data are not versioned. This further adds to the complexity as to reproduce results not only code, but data is also required. Similar to code, data also evolve.
- No model versioning or model registry is available. It is harder to reproduce a model from the past.
- Code reviews are not performed in ML. Testing is missing, such as unit or integration tests are not performed, which are commonly seen in traditional software development. The code development is limited to individual development environments, such as jupyter notebooks, on a data scientist's local workstation.
- The end product of the manual process is a single model rather than the whole pipeline.
- Collaboration between team members is a headache as it is difficult to share models and other artifacts.
- There is neither CI nor CD in the workflow as operations and development are considered as two distinct branches of the whole process.
- There is a lack of reproducibility as the experiments and the models produced (artifacts) are not tracked. There is no concept of maintenance of experiments.
- The deployed Model can not be retrained automatically, which might be required due to a number of reasons such as model staleness or concept drift. As the process is manual, deploying a new model takes a lot of time and there are less frequent releases.

- Changing one particular parameter (for example, a specific hyperparameter that impacts the dimensions in a deep neural network) changes the total pipeline and influences the pipeline stages afterward and may lack versioning or may contradict.

3.8. MLOps Workflow

Building an ML pipeline is a strenuous task. The whole pipeline is often built sequentially with the help of tools that hardly integrate. The MLOps aims to automate the ML pipeline. It can be thought of as the intersection between machine learning, data engineering, and DevOps practices [24]. Essentially, it is a method for automating, managing, and speeding up an ML model's lengthy operationalizing (build, test, and release) by integrating the DevOps practices into ML.

DevOps practices mentioned in the previous section, such as CI and CD, have helped keep the software developers and operations team in one loop. Through their continuous integration, new reliable features are added to the deployed product more rapidly. MLOps perceives the motivation from DevOps as it also aims to automate ML model development and deployment process.

In addition to integrating CI and CD into machine learning, MLOPs also aim to integrate a further practice known as CT (Continuous Training), which accounts for the variation between traditional software development and ML. As in ML, the performance of the model deployed might degrade with time due to specific phenomena such as concept drift, and data drift [31]. There is a need to retrain the already deployed model on newer incoming data or deploy a completely new model in production. This is where CT comes to the rescue as it aims to retrain and deploy the model automatically whenever the need arises. CT is the requirement for obtaining high-quality predictions all the time [32,33].

MLOps is a relatively new discipline. Therefore, there is not a fixed or standardized process through which MLOps is achieved. As there are different levels of automation with any workflow, so is also the case with MLOps, as discussed in [33]. Level-0 is the one where one starts to think about automating the ML workflow. The first aim is to achieve continuous training of the model to have a continuous delivery in production. On top of this, there should be a facility for automated data validation, code validation, experimentation tracking, data, and model versioning. This marks the completion of MLOps level-1, where an automated workflow is executed through some triggering mechanism. Furthermore, there could be more automated steps in the above workflow, e.g., deploying a production-ready pipeline itself rather than just the model that embarks the transition into the final stage or Level-2 of MLOps [33].

With the help of the first level of automation, one can get rid of the manual script-driven ML workflow that is highly prone to error. It also lays the path for a more modular and reusable structure that can be easily altered according to changes in requirements. However, at this point, new pipelines are not developed rapidly, and the development team is limited to just a few pipelines. The testing of these pipelines and their parts is also done manually. Once the tests are done, then the pipeline is handed over to the operations team. The pipeline is executed sequentially, and each part of the pipeline strongly depends on its predecessors. If the data processing part (relatively early part of the pipeline) fails, there is no need to run the rest of the pipeline. Thus, there is a need for pipeline versioning where the whole pipeline is deployed and versioned together. This is achieved in the final level of MLOps automation. Through this, many robust pipelines are managed. It promotes more collaboration between the data scientist and operations team. An essential part of the final stage is CI/CD pipeline automation to achieve a robust automated ML development workflow.

4. Workflow Stages in MLOps

Throughout the whole MLOps project, risk managers and auditors minimize model-caused risks and ensure compliance with the predefined project requirements [30]. As depicted in Figure 2, each role involved in an MLOps scenario may influence various steps

among the workflow of implementing the solution. The task of deciding which stage should be fulfilled by which actor is not so trivial as designing an ideal MLOps workflow; multiple iterations are required. For organizations that are freshly adopting MLOps, there is a need to clarify the involvement of actors in the different stages. Reasoning from this, first, a brief listing of different roles involved in MLOps is depicted. In assembling related working packages into the different MLOps phases, e.g., Data Management, ML Preparation, and ML Training and Deployment, the responsibilities of each actor are described in more detail in the following. Additionally, requirements on supportive tools of the respective phases (as further described in Section 5.1) are referenced. Finally, for each phase, we define a section as Supportive Tool Requirements, which outline the recipes or features needed to be fulfilled in that phase. Subsequently, we outline each component in detail for that particular stage.

The involved actors of MLOps workflow are as follows:

- Data Scientists are involved in various phases of MLOps, such as designing the feasibility of elementary objectives and implementing model training which outcomes the core of ML systems, a model.
- As the Domain Expert is concerned with the engineering of requirements of domain-specific undertakings, as well as validating (partial) results, this actor is irreplaceable within ML projects.
- The assurance of ingesting real-world data in ML pipelines and optimal conditions for applying them in projects is within the responsibility of Data Stewards.
- A Data Engineer is concerned with transforming unprocessed data sets into interpretive information for the specified systems.
- Software Engineers achieve the requirement of deploying value-added ML-based products.
- Operations Engineers realize the monitoring and continuous integration and delivery of the whole MLOps workflow.

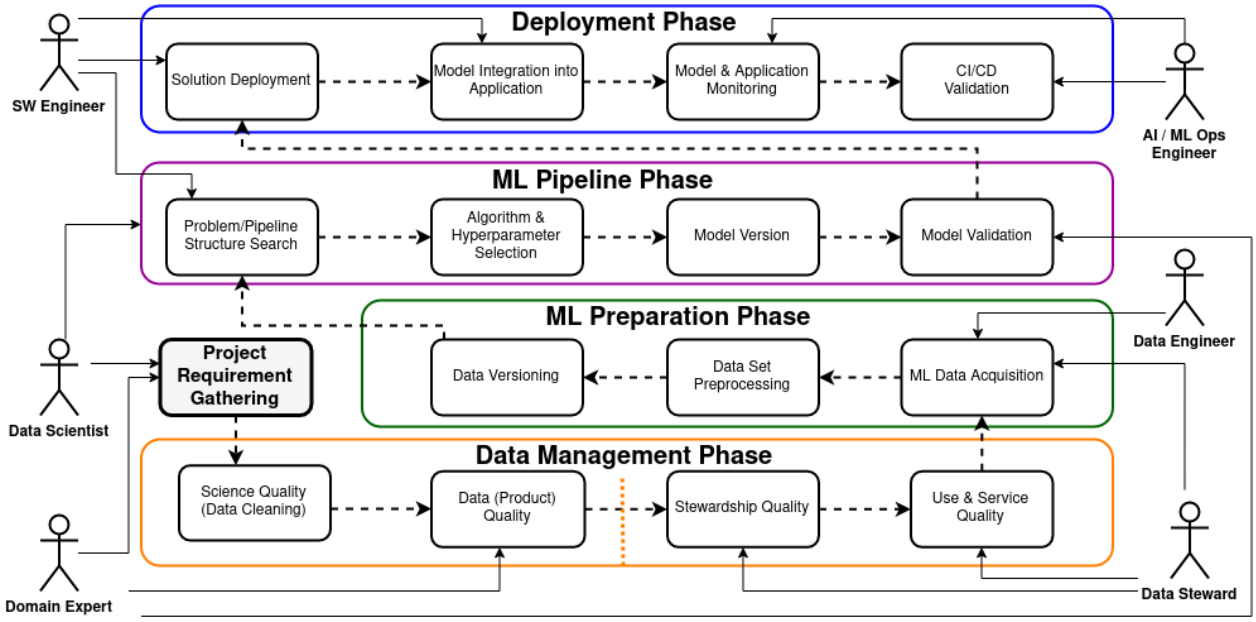

Figure 2. Common actors in MLOps and their responsibilities throughout the workflow.

4.1. Project Requirement Engineering

As the overall goal of a project is domain-specific, subject matters (e.g., domain experts) and data scientists must gather, engineer, and evaluate all necessary requirements for a successful implementation. One major task of the data scientist is to help the domain expert in framing the problem, such that it is feasible for being solved by ML technology [30].

As MLOps projects advocate for cross-expertise collaboration, engineering-specific obscurities originating at this point have to be clarified with the respective actors. In extreme cases, re-evaluating acquired concepts may be required, and implementing fine-grained discussions about potential uncertainties throughout the whole project becomes extremely important. By comparing scenarios with the problem on hand, possible workarounds, and the determination of human expertise required for solving them, a common understanding of how the ML system is applied in the end is generated. As the specifics of ML projects become more apparent within this process, the first draft of technologies required for successful implementation can be outlined. Next to data quality attributes, Key Performance Indicator (KPI)s of the resulting deployment, and the feasibility of the infrastructural design choices or demands on re-training of models must be identified. Additionally, much effort is required to fix the current business need for the ML solution. Business metrics differ significantly from the traditional ML metrics, and a high-performing model does not always guarantee that a higher business value would be generated.

4.2. Data Management Phase

As project-specific data may be restricted by domain-dependent constants, the relational integrity of attributes, the validity of historical timelines or state-dependent transitions [34], archiving overall data quality is not an easy task. Furthermore, the life cycle of data sets is associated with different quality dimensions, as described in [23]. Data quality in ML is closely related to principles of 'big data'. Quality issues should be discovered as early as possible in order to isolate and adapt faulty processes which impact subsequent utilization [17]. Subject-specific experts (e.g., Domain experts) are responsible for providing problem specific questions and goals, as well as KPIs for (data) models to create. The validation of potential (data as well as ML) models is in their area of responsibility, too [30]. As data scientists build operational ML models which address the formulated problem, the scientific quality must be assessed in tandem with the domain experts [30]. Depending on the organization's structure, there are additional involved entities and roles for data operations, for example, the Data Stewardship [35], which is responsible for supervising data during various phases of a project. With a focus on Data Quality Management (DQM) and data governance, multiple data steward-related roles were defined in early approaches in [36], distinguishing between chief, business, and technical data steward, as well as a data quality board which defines the data governance framework of the organization. Especially when planning or finishing the implementation of a project, this entity is responsible for maintaining resulting data, often represented by a Data Management Plan (DMP) [37]. In [38], the stewardship was divided into three roles, where data stewards provide guidance on data governance (e.g., data integrity, provenance, requirements, and improving quality metadata), domain knowledge, and scientific integrity (e.g., data quality and usability evaluation) is provided by scientific stewards and software, and system guidance (persisting and accessing data) is provided by technical stewards. As there is a variety of tools for supporting data stewards, as well as every other involved individual, in their domain-specific work, the best practice is to gather all tooling for each experiment in a repository, accessible by all team members [39]. Concerning the quality of actual processing data, the FAIR data principles are often quoted [39] in the context of data stewards. The definition of principles for increasing reusability of data sets describes characteristics for systems with a focus on valuable research output [40]. Even when only publishing these data sets within a project, the complete data life cycle may profit from such FAIR-implementing products, such as in [37].

4.2.1. Supportive Tool Requirements

With ingesting data into the system, the handling of the specific project-related data origins (e.g., sensors) must be integrated (DPR 1). In order to ensure applicable data, data preprocessing (DPR-2) and subsequent quality measures (DPR-3) are required. Due to

potential massive data sets, the support of managing data (DPR-4) through cockpits is recommended.

4.2.2. Science Quality

As the scientific quality of the incoming data depends on the gathered requirements of the project (see Section 4.1), data scientists and domain experts are responsible for evaluating its quality and usability [30]. From defining data accuracy and precision requirements for the intended use of information as a first critical stage, input data quality assurance, data generation quality control, and DMP are derivable during the development of input data-specific processing algorithms [23]. Next to raw data features like data types or formats, the processing of generated information (e.g., sensed or accumulated data) depends on domain-specific features like semantics, environmental influences, and statistical properties [41].

4.2.3. Data Quality

Assessing the quality of a data set is a complex and multidimensional problem [23]. According to defined production workflows, data are produced by the product, e.g., a sensor. An evaluation of the actual production workflow, the produced data quality, and a comparison of the assessed quality with similar products is carried out by a domain expert in order to pinpoint data error sources, quality assurance, compliance procedures, and data processing flowcharts [23].

There are multiple factors to the data quality dimensions as mentioned in [16], such as, for example, completeness of data in which problem-related missing values are considered and must be addressed. According to the authors of [17], completeness, data format and structure (e.g., consistency), accuracy (e.g., realistic values), and timeliness are the cases of the intrinsic quality dimension. To compute a data quality dimension score by applying the feasible metrics, other representational (e.g., interpretability or ease of understanding) or contextual quality dimensions like reputation or relevancy can also be measured. Additionally, ML-related quality dimensions like data dimensionality reduction, data heterogeneity, or data duplication are considered according to authors in [42].

Regardless of whether the data is applied in an ML pipeline, most scenarios depend upon automated data quality checks and perform them by either incremental or batch processing of the new data.

4.2.4. Stewardship Quality

In supervising rich data set metadata, its ingestion, and archiving, the overall data set quality is preserved [23] by data stewards. By ensuring metadata completeness and satisfying metadata standards and data accessibility, the data set's maturity is evaluated, and recommendations for (re)use of this iteration can be declared. This documentation of the data sets stewardship maturity evaluation and data fixation enables a subsequent data *access* of well-defined data.

4.2.5. Use and Service Quality

Dependent on the overall product problem, possible services must be selected while considering the provision of secure and stable interfaces for obtaining data sets, offering user support, or act as central feedback collectors. The data product may be reused for alienated purposes or being improved, based on user feedback in subsequent steps [23]. Additionally, the data stewards are required to prevent the leakage of sensitive data [11], which may require the input of domain experts in order to identify, for example, privacy risks in data usage concepts.

4.3. Machine Learning Preparation Phase

This collection of functions in the ML preparation phase is mainly related to classic ML preprocessing tasks. Dependent on the previously DMP, various roles must ensure

the overall data quality for the domain-specific ML solution. As data engineers are responsible for bringing the data into a ML model-consuming structure [30], support from data stewards for data ingestion is required. The overall previously defined domain-specific problem and subsequent plans for implementing the ML model also requires interaction with data scientist and domain experts.

4.3.1. Supportive Tool Requirements

By preparing data for utilization by machine learning techniques, the versioning of data sets (DPR-5) is required to reference respective data origin for a specific model. As there is often no elegant way to enhance data with domain-specific information while sensing or generating them, and many use cases require support for manually labeling specific data (DPR-6).

4.3.2. ML Data Acquisition

In traditional big data, the acquisition of data refers to the initial collection from sensors to the transport of data by interconnected networks to storage solution and a subsequent pre-processing (integration, enrichment, transformation, reduction, cleansing, etc.) [17]. Dependent on the product, not all collected data is relevant for the ML pipeline. Based on the previously declared DMP, a data engineer feeds the ML pipeline with an appropriate selection of the provided data. With support from data stewards, the optimal circumstances for obtaining problem-specific data are implemented.

4.3.3. Data Cleaning and Labeling

Next to the elimination of existing errors in the input data, procedures for feature engineering, carried out by data scientists (in cooperation with domain experts), are necessary for other domain-specific ML operations. Data cleaning can be split into three parts [43], where error detection like duplicate data, violations of logical constraints, or incorrect value recognition is the first task. Moreover, solving every detected error is a second operation, and the data imputation supplements the missing and incomplete data as the last step.

4.3.4. Data Versioning

In order to build robust and reproducible models, special attention must be payed to the versioning of every data source involved in the project. The prevention of data leakage [11], e.g., the strict separation of test, training, and validation data sets, is vital to the success of ML models When the characteristic of input data changes, the re-training of existing models, the fine-tuning of hyperparameters applied respectively may be necessary. By associating the specific circumstances, e.g., a new data version or iteration, the foundation of 'audit trails' for input data is created. In contrast to simple ML-based approaches, where a finite set of samples trains a model, specific data versions may be extended over time with new samples or features. The advantages of versioning every piece of data involved in ML projects can be summarized as tracking the evolution of data over time. As there is often a massive amount of frequently changing input data involved in ML projects, local data repositories (e.g., edge devices) often hold the actual data set. At the same time, the remote storage solutions only persist a hash of the current version [18].

4.4. ML Training Phase

This is the phase where data scientists play a significant role. These experts ensure the flexibility and scalability of ML pipelines, as well as an appropriate technology selection and efforts in model performance improvement [30]. In order to train a model until a specific goal (e.g., accuracy threshold) is reached, data scientists are responsible for choosing ML pipeline structures, algorithms and hyperparameters through model versioning and validation. They are the most expected users for the data processing phases in big data

projects [17]. Every model version candidate is evaluated, and one is chosen with the domain expert for further deployment.

Dependent on the required degree of automation in search for an ML pipeline, data scientists and domain experts are supported by different so-called AutoML techniques and tools. This area of ML was started to let non-technical domain experts somewhat configure the model training instead of having to implement the individual steps manually. In assembling different ML concepts and techniques, the development of feature preprocessing, model selection, and hyperparameter optimization are automated [44]. The process of selecting an algorithm, as well as its hyperparameters, is often implemented in one singular step and is referred to as CASH. In order to improve the performance in automated model training and hyperparameter optimization, techniques like k-fold cross-validation can be applied for early stopping the training when reaching a particular threshold [22].

4.4.1. Supportive Tool Requirements

For implementing the actual ML training, a variety of supportive tools exist. When manual model type selection (TP-1) is required, recommendations of respective hyperparameters (TP-2) support inexperienced data scientists (as well as other individuals as non-technical domain experts) in choosing well-performing model configurations. Despite the required degree of automation while training a model, tracking the whole model run (TP-3), including quality metrics applied for validation, creates an audit trail. As there are multiple ML libraries and frameworks, many with a focus on a specific type of problem, the support for integrating them into the MLOps workflow (TP-4) is often required instead of decoupling them from the pipeline. In versioning the code for training the model (TP-5), another audit trail is created for later inspection. With packing the trained model into reproducible archives (MM-1) and their persistence in a model registry (MM-2), the foundation for ML market-place-alike structures can be set.

4.4.2. Pipeline Structure Search

Dependent on the type of data (e.g., structured or unstructured input data) and the appropriate technique to solve the problem (supervised, unsupervised, or semi-supervised learning), the overall ML pipeline structure differs. As each problem probably requires its individual set of domain-specific quality demands, the regarding model performance monitoring metrics (as exemplary shown in Section 5.1.5) must be defined.

4.4.3. Algorithm and Hyperparameter Selection

The selection of the most appropriate ML algorithm for a specified problem (e.g., neural networks, vector machines, and random forest) is carried out by the data scientists. On top of this, each algorithm has parameters that must be tuned for achieving good performance. These parameters are called algorithms hyperparameters, for example, the number of layers or neutrons within a neural network. Tuning these hyperparameters and selecting an optimal algorithm is one of the most resource-intensive and tedious tasks in an ML workflow. AutoML aims to simplify this and many other manual tasks in modeling by making decisions in an automated way. AutoML aims to decrease the human effort required for building efficient ML systems. It advocates for fairness and reproducibility. It supports the data scientist such that new experiments could be tried rapidly and helps the enterprise to quickly develop new ML systems through automating much of the modeling activities. A comprehensive guide on AutoML is provided in [21]. Regarding the automation of CASH which is a sub part of AutoML, different strategies (such as, for example, generating a grid of configurations and evaluating all of them or applying random search for each hyperparameter) exist [22].

4.4.4. Model Version

ML models are fairly complex and have strong interdependency on data, framework, and modeling procedure. Model versioning is a way of keeping a track of these interde-

pendencies. It is also an essential feature through which models could be rolled back if something goes wrong in production. Due to several reasons, different models could be required at different time-frames, and versioning helps in deploying the correct version at the right time. It increases accountability and works as an essential component required for governance in ML. With storing each model iteration configuration (e.g., chosen hyperparameters, data version, and quality demands) and persisting the actual versions of trained (and well performing) models, a comparable history of solutions to specific input data is created. Furthermore, the combination of model and data version makes the sharing and re-validating results in a community easy.

4.4.5. Model Validation

Dependent on the input data generated at the foremost data management phase (see Section 4.2), as well as the type of implemented learning (e.g., supervised learning or unsupervised), the test and validation sets are applied for validating the learned model and ensuring the prevention of overfitting the model. As model performance is an indirect measurement of data quality applied in training [42], bad-performing models may be enhanced by fixing errors in the data management phase.

4.5. Deployment Phase

The deployment phase marks the most critical phase of the MLOps journey. In this phase, software engineers integrate the validated models into the respective applications and ensure the operational stability of the whole applications system [30]. As aspects of the various decisions and configurations of the whole pipeline are derivable in this integrative phase, permanent monitoring of the model, the overall application, and consuming data must be implemented by MLOps Engineers. Another role in this infrastructural deployment phase is the DevOps engineer, which is responsible for conducting, building, and testing the working system [30].

4.5.1. Supportive Tool Requirements

As the trained, validated, and registered model is ready for utilization, there are multiple application-specific and infrastructure-dependent support requirements for different deployment patterns (DP-1). As the overall system state (OMP-1) and its input data (OMP-2) will influence the overall model performance (OMP-3), constant monitoring is required. In assuring the model's quality (OMP-4), requirements for operating the complete application are finally met.

4.5.2. Solution Deployment

The sub-phase solution deployment is mainly focusing on extracted value out of the trained model. There are many potential problems and challenges in applying a trained model within existing infrastructures. Next to complications with the system-related monitoring metrics described in Section 5.1.5, various organizational or project-specific constraints, network restrictions, or missing required technologies in the ecosystem may hinder the deployment of the solution.

4.5.3. Model Integration

There are three straightforward approaches of providing an application with the overvalue of an ML model, as briefly demonstrated concerning the framework *ML.NET* by Lee et al. in [45]. Next to deploying a model inside containers and connecting them via Remote Procedure Call (RPC) to the serving system (the application respectively), the direct integration of the model execution into the application decreases custom engineering effort during deployment. Another approach is to white-box the model, where different models are represented as Directed Acyclic graph (DAG) and registered inside a serving system which is accessible via RPC or REST calls. The exchangeability of models decreases with integrating its execution within an application's source code. At the same time, the

central deployment via REST endpoints will increase the application's dependability on network connectivity and service availability.

4.5.4. Model and Application Monitoring

The constant monitoring of the whole project stack, e.g., model and application performance, as well as infrastructural circumstances and (most importantly) the data utilized by the model, is the foundation of a robust ML-based product. In utilizing metrics suggested in Section 5.1.5, many operational errors and shortcomings can be compensated in advance. With a focus on tracking the data through the pipeline operations [11], graph databases can help in managing and maintaining linkage between data objects and the respective assertions.

4.5.5. Continuous Integration and Delivery/Deployment Validation

With the goal of continuously integrating newer trained models inside the application, CI/CD Validation is the last step of the workflow in MLOps projects. To have permanent monitoring and testing of new iterations of the ML-based applications, the CI/CD validation can be applied for indicating a certain product quality and therefore is an essential component. With validating the predefined KPIs of every phase within the project, a holistic assessment of the ML-based product implementation can be derived.

5. Tooling in MLOps

In recent years, many different tools have emerged which help in automating the ML pipeline. The choice of tools for MLOps is based on the context of the respective ML solution and the operations setup [24].

In this section, an overview of the different requirements which these tools fulfill is discussed. Note that different tools automate different phases of the ML workflow. There is no single open-sourced tool currently known to us that can create a fully automated MLOps workflow. Specific tasks are more difficult to automate than others, such as data validation for images. Furthermore, some of the standard Full Reference-based Image Quality Assessment (FR-IQA) metrics are listed. These metrics could be applied for automating the data validation part. After discussing typical demands on such tools, 26 open-source tools are benchmarked according to these requirements. A ranking is provided indicating their fulfillment.

5.1. Requirements on Tools in MLOps

In the following, various requirements for MLOps are discussed, including general demands on tooling. These set of requirements form a typical recipe based on which different tools could be selected. The requirements are based on the stages introduced in the last section. By defining these requirements as the ones that the MLOps tools must address, each can be matched to one or more of these requirements. One can identify combinations of tools that cover a range of these requirements. Furthermore, formalizing the various applicable quality metrics is out of scope for the work on hand, a brief overview of image quality, system monitoring, data monitoring, and ML system monitoring metrics is given.

5.1.1. Data Preprocessing Phase (DPR)

ML models are only as good as their input data. The data collected from different sources undergo different preprocessing steps through which they are prepared to be used by an ML algorithm. There are various requirements for implementing the preprocessing steps in ML projects.

DPR-1: Data Source Handling

Integrating different data sources in a project's pipeline can be accomplished by using connectors or interfaces to load data. As every data science project depends on data sources,

handling various sources (databases, files, network sockets, complete services, and many more) is vital to the pipeline.

DPR-2: Data Preprocessing and Transformations

As input data are frequently formatted in incompatible ways, the preprocessing and transformation of data must be supported. Often, the domain-specific input data are of a structured nature like text or numbers. As the ML project is dependent on cleaned data, other unstructured formats like audio, images, or videos must be taken into consideration, too.

DPR-3: Data Quality Measurement Support through Metrics

Automated data validation is a critical component of MLOps. In order to ensure data quality for images, one can utilize reference-based Image Quality Assessment (IQA) methods. These methods require a high-quality reference image to which other images are compared. Some of these methods are listed in Table 1. Other aspects of data quality measurements are auditioning of data (unit and integration testing for data).

Table 1. Full-reference-based deterministic metrics for automated image quality assessment.

Metric	Brief Metric Description
SSIM [46]	Predict image quality by measuring the structural similarity between two images.
MS-SSIM [47]	SSIM based on modeling of image luminance, contrast and structure at multiple scales.
MC-SSIM [47]	Motion-compensated SSIM by comparing input to previous images/frames in a video.
IW-SSIM [48]	Determines the Information Content of an image and performs Weighting SSIM.
3-SSIM [49]	Three-component SSIM determines edges, textures, and smooth regions, and associates different weights for each region.
G-SSIM [47]	Gradient SSIM compares edge information between distorted and original images.
NQM [50]	The Noise Quality Measure models distorted images by using a linear frequency distortion with additive noise injection.
VIF [51]	Visual information fidelity is calculated by relation of Distorted Image Information to the Reference Image Information.
FSIM [52]	Feature Similarity Index maps features and compares them to the original image.
GMSM [53]	Gradient Magnitude Similarity Mean makes use of average pooling on GMS maps.
GMSD [53]	Gradient Magnitude Similarity Deviation using pixel-wise GMS captures local image information. The standard deviation of the GMS-map is the final image quality index.
UQI [54]	Compute an 'Universal Quality Index' by loss of correlation, luminance distortion.
MAD [55]	The most apparent distortion metric utilizes contrast sensitivity, local luminance, and contrast masking for high-resolution images. For low-resolution images, changes in local statistics between the originals image sub-bands are compared.
DSCSI [56]	Directional Statistics based Color Similarity Index can handle chromatic and achromatic distortions.
VSI [57]	Visual Saliency-Induced Index uses 'saliency map' as a quality feature and weighting function to characterize the importance of local regions.

DPR-4: Data Management

Managing data sets in a centralized cockpit is an obvious requirement of MlOps. In visualizing diverse aspects of gathered information (for example, the distribution of specific attributes values or the alteration of a variable over time), the respective user is supported with a quick overview of potential problems or errors in the data set.

DPR-5: Data Versioning

Data versioning is an essential requirement for implementing MLOps, similar to the importance of code versioning in DevOps. Without data versioning, reproducibility cannot be achieved in ML. The approaches for data versioning can be divided into two sections, i.e., git-style alike and time-difference-based. The git-style approaches track immutable files but cannot store the differences between them.

DPR-6: Data Labeling

The labeling of data is a mandatory task in supervised learning scenarios. When manually preprocessing such data sets, several tools can support the respective actor (e.g., data scientist and domain expert) in extending the various samples with a target label (e.g., the attribute to be predicted when executing the model).

5.1.2. Model Training Phase (TP)

Training of ML models is a resource-intensive, complex, and iterative process. ML is capable of offering multiple correct solutions for a single problem, unlike traditional technologies that offer only one perfect solution [58]. In this section, we define requirements based on challenges faced during the training of an ML model.

TP-1: Model Type Selection

In providing a user with the choice of defining a model type, as the selection between forest tree or naive Bayes, the training is made configurable. When offering the basis for the solution of multiple problem types, such supportive tools for model training may also be applied for further operations within an organization.

TP-2: Hyperparameter Tuning

Once an algorithm for solving a well-defined problem is chosen, its configuration (e.g., the model's hyperparameters) must be selected. Concerning automating this technically sophisticated task, various AutoML approaches can lighten the data scientists' workload by applying CASH techniques.

TP-3: Tracking

ML is intrinsically approximate, and the development process is iterative. As the number of iterations increases, the information necessary for reproducing that experiment also grows. Tracking in ML is a process through which all modeling-related information, i.e., the algorithm, its hyperparameters, and the observed performance based on some metrics, could be saved automatically in a database. This collection of meta-information helps compare algorithms, runs, and performance metrics.

Tracking the training runs, configured hyperparameters, and the corresponding model quality metrics are the overall foundation of the model's quality. Especially when implementing multiple different ML models as a solution to a problem in parallel, comparing results of measurements may help in selecting feasible model candidates. ML experiment tracking tools are an essential part of a data scientist tool kit.

TP-4: Integration Friendly

As there is a multitude of different ML frameworks (e.g., Pytorch, Scikit-Learn, or Tensorflow), each with a focus on a specific ML technique, concept, or implementation details, the integration of arbitrary frameworks in the MLOps workflow enables diverse technical approaches for problem-solving.

TP-5: Code Versioning

The failure of ML systems does not create explicit errors, and pointing to the source of the error is quite a difficult task. To revert quickly back to the previous working solution, versioning is necessary. As in DevOps, code versioning is also an essential part of MLOps. With versioning, the applications code, advantages in classic software engineering tasks (branches, rollbacks, etc.) are realized. The demand for such technologies increases with the complexity of the application and the actors involved in its implementation. Code versioning stimulates successful deployments and reduces the chaos which surrounds the development process.

5.1.3. Model Management Phase (MM)

Model management is also an essential principle of MLOps. As the number of ML models created in an organization increases, it becomes more challenging to manage them. ML model management is required so that new models can be consistently deployed, and if something goes wrong, then they can also be reverted easily. In this section, we define requirements based on challenges faced during the management of ML models.

MM-1: Model Packaging

As the deployment of ML applications is dependent on the respective technologies, the packaging of trained models must be considered. Depending on the MLOps projects' motivation and objectives, sharing resulting models within a scientific community may be expected.

MM-2: Model Registry

The goal of the model registry is to store every model along with all of its versions and metadata. Without a model registry, the model serving tool cannot process automatic requests of querying the models. The registration of models within a central registry helps in coordinating the selection and deployment of specific versions. It helps in governance and offers better visibility while decreasing the time to production for already saved models.

5.1.4. Model Deployment Phase (DP)

As the deployment of the ML model is organization-specific, restrictions in deployment options of a model may hinder the utilization of supportive tools.

DP-1: Support for Different Deployment Patterns

Concerning the physical location of where the prediction should be executed, different deployment strategies are possible. Next to offering the prediction as a service, deploying the edge or inside a container structure or cluster is thinkable.

5.1.5. Operations and Monitoring Phase (OMP)

Different monitoring metrics are feasible depending on the MLOps project's structure and approach to solve a specific problem. As the model performance depends upon input data, as well as the overall system infrastructure, different observation aspects and strategies for the overall status of the ML environment are summarized in the following.

OMP-1: System Monitoring (SM)

The constant monitoring of the deployment infrastructure (for example, the systems latency, throughput, or server load) is the foundation of counteracting deployment-related errors. Different metrics for monitoring the overall system utilized for ML tasks are depicted in Table 2.

Table 2. Metrics for system monitoring.

Metric	Remarks
Serving latency	The metric measures the response time. Low latency is one of the critical aspects of a good user experience. In order to create an expressive measurement, the responses of every tier must be taken into consideration. Next to the average time to answer a request, other aspects like perceived page load time may influence the metering [14].
Throughput	Average predictions returned in one second. High throughput is representative of a healthy system. Additionally, measurements regarding the throughput could be impacted by additional metrics, e.g., total hits per week, average hit size, or the occurring error rate [14].
Disk utilization	Monitoring free disk space helps prevent data loss and avoid server downtime. The statistics of the respective server's filesystem may also help in determining and understanding the applications health status [13]
System's uptime	System uptime is an important measure of the reliability of the system. It accounts for the availability of the service. Therefore, this performance metric type is especially useful when serving web-based applications [14].
CPU/GPU usage	Monitoring GPU/CPU usage helps identify whether some inputs result in longer computation time. This information is beneficial for tuning the model for a better user experience. While fine-tuning the capacity and configuration of infrastructure, these resources may be resized in order to fit the required serving capability [14].
Number of API calls	While serving the model to many users, monitoring the number of API calls is a good metric for approximating the maximum load and systems availability.

OMP-2: Input Data Monitoring (IDM)

When not permanently monitoring the incoming data of an application, error factors like data or concept drift are easily overlooked, and impacts are to be expected in the subsequent model utilization. Different aspects of monitoring the respective data sets are overviewed in Table 3.

Table 3. Metrics for input data monitoring.

Metric	Remarks
Data schema	This metric aims to check whether the data schema in the inference stage is identical to what was there during training. It is an important measure of data integrity. By assuring such referential integrity, a certain degree of data quality, as well as assumptions on data completeness can be observed [15].
Data consistency	The metric checks if the data encountered in production is within the expected range. It could also be used for catching data type or format errors. By comparing the amount of records to previously known states of the dataset, another consistency factor can be measured [15].
Data quality	The metrics checks if the quality of data is similar to what the model was trained on. If the quality is changed, then that will have a direct effect on the model's performance. Therefore, the different data quality dimensions must be assured in every stage of the data lifecycle [17].
Data drift	The metric aims to examine changes in the distribution of data. It checks whether or not the input data's statistical structure (changes in input feature space) has changed over time.
Outliers	Many algorithms are sensitive to outliers and perform poorly on such data points. Outliers could bring additional knowledge about the use case and therefore should be investigated in detail. When comparing the incoming information to a previously known reference or mathematical rule, the validity of data can be measured [15].
Training/Serving skew	The aim is to check if there is a mismatch of data between data acquired for training and what is received in production.

OMP-3: Model's Performance Monitoring (MPM)

As patterns in the incoming data change dramatically over time, the model will not be able to perform as expected (which is referred to as model staleness) and must be re-trained. Therefore, having a close look at the model's performance helps in preventing decreasing effectiveness of models. The metrics summarized in Table 4 point out various problems which have to be evaluated before and during deployment for production.

Table 4. The different metrics that can be used for monitoring ML systems.

Metric	Remarks
Output distribution	The distribution shift of the predicted outputs between data from the training and production phase indicates model degradation. Metrics such as the Population stability index or Divergence index could help in alerting when such shifts happen. Especially in time-series-based datasets, this metric is essential in order to determine the credibility of existing models on the new data [19].
Model's performance	This particular metric is dependent on the task at hand. It is useful for checking whether the current model is good enough or not. Examples are Auc score, precision, recall, FPR, TPR, and Confusion Matrix for classification tasks. Such metrics are difficult to calculate as there might be long latency between the output and true label. In general, a model's performance can be evaluated by its accuracy (e.g., right predictions), gains (comparison to other models), and accreditation (credibility of resulting predictions) [19].
Feature importance change	The model's performance might not degrade, but there could be a change in feature importance based on which the predictions are made. This could be an indication of drift and possibly a point where data should be collected again for retraining.
Concept drift	Concept drift is a major source of model performance degradation. It represents the pattern which the model has learned in the past is not valid anymore. For a comparative discussion of active and passive concept drift detection in distributed environments, we refer to the work in [20].
Bias/Fairness	The trained model could be biased towards one class or group. Accuracy difference (AD) can be measured for each class to check for model bias. Another possibility is that the model can be evaluated on specific slices of data for obtaining an in-depth review of the model's performance.
Numerical stability	Numeric values that are invalid or wrong are possibly learned during model training. These do not directly trigger any explicit errors but can produce significantly wrong predictions in production. Checking for numerical stability ensures robustness.

5.1.6. General Requirements

Other than the ML-specific requirements, there are some additional elements that could play an important role in the selection of tools. In this section, we define four key points which are also used commonly for other software domains.

GR-1: Scalability

The scalability of supportive tools for data science projects is a basic requirement. Depending on the specific task, various aspects like the required processing power, storage amount, or service latency must be flexible.

GR-2: APIs

By controlling a tool via its served APIs, many tasks can be realized programmatically, potentially strengthening the robustness of the MLOps pipeline by automating specific tasks and eliminating errors of manually utilization or configuration of tools.

GR-3: Tool Maturity Level

The reliability of tools in production environments is a vital requirement, as errors may dramatically impact business goals. Therefore, a maturity assessment of potential tools (e.g., well-tested, widely spread utilization, etc.) should be carried out beforehand in any case.

GR-4: User Friendliness

As many actors are active within the MLOps workflow, non-technical personal must be supported by user-friendly tools (e.g., self-explaining or intuitive user interfaces).

5.2. Tool Comparison

In this section, different open-sourced tools for MLOps that have emerged in the recent past are compared and overviewed. These tools are benchmarked in Table 5 according to the requirements mentioned in the previous section by additionally giving a ranking of how well they fulfill those requirements. The ranking is as follows:

- o: Feature is missing.
- +: Feature is present but does not fulfill the complete requirement or is not user friendly.
- ++: Feature is present and is also easy to integrate and user friendly.

A short overview of existing tools could help in identifying key markers based on which an individual can decide whether to choose a particular tool or not. In the following subsections, we discuss the critical remarks for each of the tools, which we find essential for the selection for creating a successful MLOps workflow.

5.2.1. MLflow

MLflow [59] is a Machine Learning Platform that can be used for partial automation for small- to medium-sized data science teams. The tool has four parts—Tracking, Model, Projects, and registry—targeting a specific phase of the ML lifecycle. The feature of User management is missing from the tool. It is also not fully customizable (it cannot group experiments under one architecture).

5.2.2. Polyaxon

Polyaxon [60] is a tool that focuses on two things, i.e., first, it promotes collaboration among ML teams, and second, it focuses on the complete life cycle management of ML projects. Unlike MLflow, it Deals with user management and offers code and data versioning support. The platform is not fully open-sourced, and it needs Kubernetes. Therefore, it is not fully architecture agnostic. It provides an interface to run any job on a Kubernetes cluster. The platform requires greater setup time and might be unnecessary for small teams.

5.2.3. Kedro

The main focus of kedro [61] is pipeline construction and providing a basic project structure for ML projects (scaffolding). It offers the capabilities of defining pipelines by using a list of functions, and it also provides inbuilt file and database access wrappers. It also offers code versioning functionality with pipeline visualization (with an extra python package, 'Kedro-Viz'). However, with the pipeline visualization feature, there is no functionality to see real-time progress monitoring.

Table 5. A comparison of supportive tools for the different MLOps phases.

Tool Name	Introduction/ Use Case	Data Preprocessing	Model Training	Model Management	Model Deployment	Operations & Monitoring	General Requirements
Mlflow (v1.14.1) [59]	An open-sourced tool kit for ML experiment tracking, registry, packaging and lifecycle management.	o	+ TP-3 TP-4	+ MM-1 MM-2	+ DP-1	o	+ GR-2 GR-3 GR-4
Polyaxon (v1.9.5) [60]	An enterprise-grade platform for building, training, and monitoring large-scale deep learning applications.	+ DPR-5	++ TP-2 TP-3 TP-4 TP-5	++ MM-1 MM-2	+ DP-1	+ OMP-1	+ GR-1 GR-2 GR-3
* Kedro (v0.17.3) [61]	A python-based tool for creating reproducible, maintainable, and modular data science code. It offers CLI and UI to visualize data and ML pipelines.	o	+ TP-4	+ MM-1	+ DP-1	o	+ GR-3 GR-4
TFX (v0.30.0) [62]	TFX is an end-to-end platform for deploying production ML pipelines.	+ DPR-2 DPR-3 DPR-4 DPR-5	+ TP-3	++ MM-1 MM-2	+ DP-1	+ OMP-2 OMP-3	+ GR-1 GR-3
* ZenMl (v0.3.8) [63]	A lightweight tool for creating reproducible ML Pipelines. It provides the ability to shift between cloud and on-premises environments rapidly.	+ DPR-1 DPR-5	+ TP-3 TP-4 TP-5	++ MM-1 MM-2	+ DP-1	o	+ GR-2 GR-3 GR-4
H20 [64]	A fully open-source, distributed in-memory machine learning platform with linear scalability.	+ DPR-1 DPR-2	+ TP-1 TP-2	+ MM-1	o	o	GR-1 GR-2 GR-4
* Kubeflow (v1.3.0) [65]	It is a Kubernetes-native platform formed out of a collection of different components that focuses on orchestrating, developing, and deploying scalable ML workloads. Comes with a GUI and CLI.	+ DPR-2 DPR-5	++ TP-1 TP-2 TP-3 TP-4 TP-5	+ MM-1	+ DP-1	+ OMP-1	+ GR-1 GR-3

Table 5. *Cont.*

Tool Name	Introduction/ Use Case	Data Preprocessing	Model Training	Model Management	Model Deployment	Operations & Monitoring	General Requirements
* Flyte (v0.15.0) [66]	A container-native workflow orchestration platform for building complex, concurrent, scalable, and maintainable workflows for data processing and machine learning. It comes with an intuitive UI, CLI, and REST/gRPC API.	o	+ TP-5	++ MM-1 MM-2	+ DP-1	+ OMP-1	++ GR-1 GR-2 GR-3 GR-4
* Airflow (v2.1.1) [67]	It is a general task orchestration open-source workflow management platform.	+ DPR-1 DPR-2	o	o	o	+ OMP-1	+ GR-1 GR-2 GR-3
DVC (v2.3.0) [68]	A command line tool that mimics Git-like format and is used mainly as a data versioning tool. It is completely free, light-free, and offers easy integration.	+ DPR-1 DPR-5	+ TP-3 TP-5	+ MM-2	o	o	++ GR-1 GR-2 GR-3 GR-4
Pachyderm [69]	A tool designated for data versioning, creating end-to-end pipelines, and creating data lineage. It comes in three different versions: Community Edition (open-source), Enterprise Edition, and Hub Edition.	+ DPR-1 DPR-2 DPR-5	+ TP-5	o	o	o	+ GR-1 GR-3
Quiltdata (v3.2.1) [70]	Data versioning wrapper tool of AWS S3. It helps to package data and create different versions of it inside an S3 bucket. There are three versions available: free, limited, and enterprise.	+ DPR-2 DPR-3 DPR-4 DPR-5	o	o	o	o	+ GR-1

Table 5. *Cont.*

Tool Name	Introduction/ Use Case	Data Preprocessing	Model Training	Model Management	Model Deployment	Operations & Monitoring	General Requirements
Great Expectations (v0.13.21) [71]	An open-source python framework for profiling, validating, and documenting data.	+ DPR-3	o	o	o	+ OMP-2	+ GR-3 GR-4
GitLFS [72]	An open-sourced project which works as an extension to Git for handling large files.	+ DPR-5	o	o	o	o	+ GR-2 GR-3 GR-4
** CML (v0.5.0) [73]	A CLI tool (a library of functions) focused on Continuous Machine learning.	o	o	o	o	o	+ GR-4
** Github actions [74]	A functionality of GitHub that helps in automating software development workflows. Not free of charge for private repositories.	o	o	o	o	o	+ GR-4
** CircleCI [75]	CircleCI is a cloud-hosted platform for creating CI/CD workflows. Not free of charge for private repositories.	o	o	o	o	o	+ GR-1 GR-2 GR-3
** GoCD [76]	Open-source Continuous Integration and Continuous Delivery system.	o	o	o	o	o	++ GR-1 GR-2 GR-3 GR-4
Cortex (v1.9.0) [77]	A multi-framework tool with CLI for deploying, managing, and scaling ML models.	o	o	o	o	+ OMP-1	+ GR-1 GR-2

Table 5. *Cont.*

Tool Name	Introduction/ Use Case	Data Preprocessing	Model Training	Model Management	Model Deployment	Operations & Monitoring	General Requirements
Seldon Core [78]	A framework that simplifies and accelerates ML model deployment on Kubernetes on a large scale.	o	o	+ MM-1	++ DP-1	+ OMP-1 OMP-2 OMP-3	+ GR-1 GR-2 GR-3
BentoML [79]	A flexible, high-performance framework for serving, managing, and deploying machine learning models. The tool has both CLI and Web UI.	o	o	++ MM-1 MM-2	++ DP-1	+ OMP-1	+ GR-2 GR-3 GR-4
Prometheus [80]	A toolkit for monitoring systems and generating alerts.	o	o	o	o	+ OMP-1 OMP-2 OMP-3	+ GR-2 GR-3 GR-4
* Kibana [81]	A framework that is an essential part of the ELK stack (Elasticsearch, Logstash, and Kibana). Kibana is used mainly for visualizing machine logs.	o	o	o	o	o	+ GR-1 GR-3
* Grafana [82]	A tool that can be used to visualize and analyze any machine-readable data.	o	o	o	o	o	+ GR-2 GR-3 GR-4
Label Studio [83]	An open-sourced web application platform for data labeling, which offers labeling solutions for multiple data types such as text, images, video, audio, and time series.	+ DPR-6	o	o	o	o	+ GR-2 GR-3
Make sense [84]	A browser based platform for labeling images.	+ DPR-6	o	o	o	o	+ GR-1 GR-3 GR-4

* Indicates that tool fulfills other requirements of the ML lifecycle such as pipeline orchestration or visualization. ** Indicates fulfillment of CI/CD tools for the ML lifecycle.

5.2.4. TFX

TensorFlow Extended (TFX) [62] is an end-to-end platform for deploying production-ready ML pipelines. The pipeline is formed as a sequence of components that implement an ML system. To benefit from TFX, one has to use the TF framework libraries for building individual components of the pipeline. The pipeline can be orchestrated using common orchestrators like Apache Beam, Apache Airflow, and Kubeflow pipelines. It has higher integration costs and might not be ideal for small projects where new models are required less frequently.

5.2.5. ZenML

ZenML [63] is a lightweight tool for creating reproducible ML Pipelines. It provides the capability to shift between cloud and on-premises environments rapidly. It is focused on pipeline construction formed through a combination of steps by using Apache Beam. It has a UI for visualizing pipelines and comparing different pipelines in one Zenml repository. However, it does not offer support for scheduling pipelines currently. It has powerful, out-of-the-box integrations to various backends like Kubernetes, Dataflow, Cortex, Sagemaker, Google AI Platform, and more.

5.2.6. H2O

The H2O platform [64] is a part of a large stack of tools where not every tool is open-sourced. It comes with a web UI. It is more like an analytics platform, but it is also regarded as a library due to the different APIs it offers. The tool is used mainly for running predefined models with automl capabilities, but it does not give the freedom to integrate state-of-the-art custom models. It has leading AutoML functionality. It is Java-based and requires Java 7 or later. Additionally, it lacks the capabilities of model governance.

5.2.7. Kubeflow

Kubeflow [65] is a specialized tool for orchestrating ML workflows. One disadvantage of the tool is that it has high competency with Kubernetes and therefore is more difficult to configure. The main goal of the tool is to make ML lifecycle and components interaction (workflow manager) easy on Kubernetes.

5.2.8. Flyte

The focus of this tool is on ML and data processing workflow orchestration. The Flyte [66] platform is modular and flexible by design. It is directly built on Kubernetes and therefore gets all the benefits that containerization provides but is also difficult to configure. It uses "workflow" as a core concept and a "task" as the most basic unit of computation. It has grafana [82] and prometheus [80] integration for monitoring. It can generate cross-cloud portable pipelines. There are SDKs for Python, Java, and scala.

5.2.9. Airflow

Airflow [67] is a workflow management platform that offers functionalities for managing and scheduling tasks (depicted as DAGs). Airflow is not limited to Kubernetes like the kubeflow tool; rather it is designed to be integrated into the Python ecosystem. It is not intuitive by design and has a steep learning curve for new users. One can define tasks in python and orchestrate a simple pipeline. It is also quite difficult to integrate Airflow for ML projects which are already under development.

5.2.10. DVC

DVC is a Version Control System for ML Projects [68]. The tool is capable of tracking models, datasets (including large dataset 10–100 Gb vs. 2 Gb limit on Github), and pipelines. It is cloud- and storage-agnostic, i.e., the datasets can be stored, accessed, and versioned locally or on some cloud platform. It is git compatible and can track experiments but does not offer a dashboard. It works by creating the file '.dvc' in the project root, where meta

information such as storage location and format of the data is stored. For unstructured data, the changes are tracked as new versions by themselves, which requires high storage capacity. It is a very lightweight tool which offers great features with the minimum effort required for integrating it into any Gitflow workflow.

5.2.11. Pachyderm

Pachyderm [69] is another tool that runs on top of Kubernetes and Docker. It helps configure continuous integration with container images. It has a git-like immutable file system. It has a steep learning curve due to dependency on the Kubernetes cluster if using the free version. Compared to DVC, it offers many options for data engineering, such as creating a data lineage by tracking the sources of data. The web dashboard is not available in the free version.

5.2.12. Quilt Data

If AWS services are being used for the ML workflow then Quilt data [70] is an excellent choice for data versioning. Data quality testing support is also available. Sharing data between different projects is easily accomplished. The tool consists of a Python client, web catalog, and a backend to manage data sets in S3 buckets.

5.2.13. Great Expectations

Great Expectations [71] is a tool for data validation and data profiling. It can be easily plugged and extended with many frameworks. It offers integration support for many data sources including MySQL, SQLAlchemy, and others. The main purpose of this tool is to do data quality assessment through automated testing, but it is limited to tabular data only.

5.2.14. GitLFS

It is an extension created by Atlassian for Git. It introduced the concept of tracking pointers instead of the actual files. Unlike DVC, Git-lfs requires installing a dedicated server. The servers cannot scale as DVC does. The framework is used for storing large files, but with GitHub, that limit is up to 2 GB (limitation of Github as Github has a 2 GB limit of the file size with Git LFS).

5.2.15. CML

The tool is for implementing CI/CD/CT functionality in ML projects. It is ideal if one is using GitFlow for data science, i.e., using git ecosystem and DVC for ML development lifecycle. Every time some code or data is changed, the ML model can be automatically retrained, tested, or deployed. It relies on GitHub actions and GitLab CI but offers an abstraction on the top for better usability. It is intended to save model metrics and other artifacts as comments in GitHub/Lab.

5.2.16. GitHub Action

GitHub actions [74] are not a tool but a new feature for setting up CI/CD workflows within the GitHub platform. Actions are accessible for every open-source GitHub repository. The private repository has 2000 build minutes as a limit for free.

5.2.17. CircleCI

CircleCI [75] is a cloud-hosted platform for creating CI/CD workflows that is not free of charge for private repositories. It is an alternative to CML but with more functionality. It has a dashboard and can generate detailed reports. It is a complete continuous integration system. The tool was originally developed for DevOps and not intended for ML purposes.

5.2.18. GoCD

The GoCD tool is focused on Continuous Delivery (CD). It can be used for tracing (visualizing) the changes in the workflow and also for creating one. It has native Docker

and Kubernetes support. For the open-source version, you have to set up and maintain the GoCD server (acting as central management endpoint). Different options for the automation of triggers are also available.

5.2.19. Cortex

The main purpose of the tool is ML Model serving (as a web service) and Monitoring. It is built on top of Kubernetes and has the capability of autoscaling. It requires a lot of effort for setting up and currently supports AWS only (therefore, the tool can be seen as an open-source alternative to Amazon's Sagemaker). Cortex includes Prometheus [80] for metrics collection and Grafana [82] for visualization.

5.2.20. Seldon Core

Seldon core [78] also has a strong dependency on Kubernetes. It can be thought of as a wrapper around the model. It comes with multiple pre-packaged Inference Servers. It is a language-agnostic framework with support for many ML libraries. The tool allows the creation of more complex inference graphs where multiple model artifacts are containerized into separate reusable inference containers that are linked together.

5.2.21. BentoML

The framework is focused on building and shipping high-performance prediction services, i.e., models serving the purpose of solving specific problems. With just a few lines of code, the model can be converted into a production REST API endpoint. Similar to MLflow models, it offers a unified model packaging format. It acts as a wrapper around the ML model for converting it into a prediction service. This approach is also followed up by the tool Seldon core. It offers an extensible API. There is support available for almost every major ML framework (Pytorch, Tensorflow, etc.). It currently does not handle horizontal scaling, i.e., spinning up more nodes for prediction service when required.

5.2.22. Prometheus

It is a metric-based monitoring and alerting system which scraps metrics and saves them to a database as time-series data. Applications do not send data to Prometheus, but it is the tool that pulls data. Each target is scrapped at a fixed interval. It is highly customizable and modular. The only disadvantage is that with time, managing the database and the server becomes difficult. The tool is mostly used together with Grafana [82].

5.2.23. Kibana

Kibana is a data visualization and exploration tool. The user has to connect to Elasticsearch (a popular analytics and search engine). It can be run on-premise or on cloud platforms and has options for creating and saving browser-based dashboards. The tool has a steep learning curve for new users, especially for the initial setup. If one does not use Elasticsearch, then the tool is of no use.

5.2.24. Grafana

Grafana [82] is a metric-based tool that supports various data sources such as the Prometheus database, Elasticsearch, etc., generally used with Prometheus. It comes with an in-built alert system. It can be deployed on-premises, but this increases maintainability. There are fewer options for customization of the dashboard by default and require installation of extra plugins, which may increase complexity.

5.2.25. Label Studio

Lable Studio [83] is a data labeling tool. The interface of the tool is not pre-built, but it needs to be configured manually, similar to any typical web-based application. The platform offers SDKs for multiple frameworks for easy ML integration.

5.2.26. Make Sense

Make Sense is a free-to-use online tool for labeling images. No installation is required for using the tool as one can directly access it from the browser. The main focus is on annotating images for the object detection and image recognition task.

5.3. Selection of Tools

In the previous section, different tools are compared and a short description of each tool is provided. In general, selecting a tool depends not on the use case or domain, but on what stage of the workflow in Figure 2 is automated by the tool. Similar to an ML model, building an ideal MLOps workflow for the first time requires multiple iterations. Each company faces unique challenges when building an ML system, even though life cycle development phases remain the same.

The above table does not showcase the solutions required for developing newer ML models as the tools for modeling are mostly frameworks or libraries such as Pytorch and Keras. Furthermore, there are standalone tools available for some functionalities, such as Prometheus for monitoring, but most tools offer a large variety of functionalities. This might cause extra redundancy and lead to unwanted complications. Furthermore, tools are developed for different scalability requirements. Therefore, there is no single best tool for one particular phase for all the organizations.

Another vital point to keep in mind when selecting tools is the integration support a tool offers. Some tools have their APIs only in python. In contrast, some offer support for JAVA and R. The support of different intermediate libraries required for model building is also an essential factor. Additionally, due to the lack of interactions between the production and development teams, some extra complications might occur in the future. The production team needs to use the tool stack, which will be in sync with the development team. For example, if the development team chooses a platform such as TFX, then the deployment team is somewhat limited to the tf serving compatible solutions. There is a high dependency on the overall workflow as the construction of the pipeline occurs sequentially. In summary, when getting started with MLOps, the focus should not be on automating everything at once; instead, focus on individual stages; try to choose a tool that offers flexibility and has support for other stages also. Achieving an ideal workflow might require few iterations, and therefore creating a list of tools that every department agrees on is a must. These tools, like chains, act as the blueprint for the whole workflow, and spending extra time initially identifying exact needs is beneficial in the longer run.

In order to use the table presented in this section, one should first think about the requirements and then select the tools that fulfill those requirements. For example, if tracking functionality is required, then one can choose MLflow or DVC. However, the selection of the optimal tool is not a straight word task. As mentioned in the above descriptions, some tools have a dependency on Kubernetes, and setting up a Kubernetes cluster is an expensive operation, especially for companies that do not need frequent model updates. Organizations mostly have some frameworks in use already, and selecting a tool compatible with the already in use platform is also essential. The goal should be to fulfill the requirements, decrease the complexity, and aim for automation in smaller steps.

6. Exemplar Workflow: Partially Automating Object Detection with MLOps Tools

In this section, an example of basic level-1 MLOps workflow as discussed in Section 3.8 for "Defect detection on objects with deep learning" is shown by coupling some of the tools from the previous section. This workflow aims to showcase how we can combine open-source tools for a use case where newer models are not required frequently and there is no CD. Such an example could be valuable for people in the early stages of ML development. Most organizations already have code versioning tools such as Git and GitHub in their tool stack. This example extends the push-based development process, which has already been adopted in the industry. With fewer integrations, even the older notebooks and data science code can be reused for designing such a simple workflow.

Designing an automated MLOps workflow follows the semi-identical recipe for any domain. What changes from one domain to another are metrics involved in data quality assessment, model evaluation, and system evaluation stages. For example, when working with object detection using images, one may use IOU as a metric for model evaluation. However, the same metric cannot be used for a natural language processing task. The requirements on which the different tools are benchmarked are chosen so that the choice of the domain becomes irrelevant. Even if an MLOps workflow is designed using the tools mentioned in Table 5, the workflow will still have to be tweaked to fit a particular use case.

Concisely, the MLOps workflow for Image Processing mainly with data-driven approaches follows a similar pattern as explained in Section 3. However, it has more complexity when compared to workflows for structured datasets. The reasons for this high complexity are as follows:

- High dimensionality of the data involved.
- Interpretation and understanding of visual data is a complex task.
- Image data sets are generally enormous which require extra storage resources.
- Training of data-driven model for image processing is a time-consuming task that requires more extensive computational resources.
- The quality assessment of images is a challenging task. The image quality is dependent on various factors such as external sources (e.g., poor lighting conditions) or the camera's hardware itself.

A suitable starting point for developing the workflow is thinking of a manual pipeline for object detection and automating each stage individually. From the manual pipeline shown in Figure 1c, we need to automate five stages, i.e., Data analysis, Data preparation, Model training, Model testing, and, finally, deployment. The particular tools which automate the stages mentioned above are listed below.

- Data analysis: Git and DVC
- Data preparation: GitHub actions
- Model training, Model testing, and deployment: MLflow

We only focus on the design for simplification and overlook the reasoning for selecting these particular tools. Additionally, in this example, there is no monitoring of the model or the workflow. The model serving is just limited to saving ML models as a python function at a fixed location. Furthermore, these tools do not require any significant additional installations, and most of them are already commonly used for software development. This ensures reusability, and in order to automate an old model training script, one only has to integrate MLflow API and set up a new DVC repository.

Regarding the automation level of a workflow, also note that an MLOps workflow can be distinguished based on the maturity level of the data science process followed in an organization and also on how many end users interact with the ML model. A lower maturity level represents a small team working on data science with manual ML workflow execution (in most cases, an individual data scientist). In such scenarios, a model is not deployed to multiple users, and there is no monitoring of the deployed service. The data science team hands over the trained model to the operations team for further usage, like its integration into the target application.

Interestingly, most organizations start with a data-centric approach and then work their way to a model-centric and, then, pipeline centric approach when adopting MLOps [29]. The data-centric approach entails that initially, some models are created to check whether or not business value can be created from the data gathered at the organization. This is the first attempt at applying ML to data (low maturity level) where simple modeling approaches are preferred. Once the proof of concept is ready, the organization shifts to more complex state-of-the-art algorithms for solving the problem at hand (increasing maturity level). The final stage of this characterization is the pipeline-centric approach which represents that an organization has ML models which are critical for generating business value and are looking for ways to scale them for achieving continuous development (highest level of maturity).

6.1. Simple MLOps Workflow

For the use cases where new models are not frequently deployed, and deployment happens on an edge device rather than as a prediction service, even simpler Level-1 workflow might be the best fit. In such cases, automation is required for the modeling part but pipeline versioning is not a must.

The first step towards adopting MLOps for such a use case is automating the process of training a new model as soon as new data is available for production. This involves running the ML pipeline automatically with steps such as automated data quality assessment and model testing. The target problem for this workflow is defect detection on objects using images with deep learning. Anomaly detection is a major part of industrial sectors where identification of defects or frauds is widely implemented [85]. An extensive overview of detecting anomalies with deep learning is reported in [86]. However, in a data-centric approach, the selection of the best algorithm is not the focus of the process. For this purpose, *YOLOv5*, an already implemented deep neural network is selected. The implementation is provided by Ultralytics as an open-source project [87]. YOLO algorithms are commonly used for real-time object detection with images.

The goal of this workflow is to perform CT with automated hyperparameter optimization for an object detection problem. This includes the functionality of experimentation tracking, artifacts logging, and model registration. For each run, the user-defined metrics and the resulting model are logged in an SQLite database. A summary of tools used for this workflow is as follows:

- Git: for code versioning.
- GitHub actions: for triggering automated data validation and other steps of pipeline.
- DVC: for data versioning.
- MLflow Tracking: for tracking experiments and comparing runs.
- Hyperopt: for automated hyperparameter optimization.
- MLflow Projects: for packaging code and environment dependencies.
- MLflow Models: for storing the model as a python function.
- MLflow Registry: for managing and annotating models.

The workflow follows a Gitflow design for continuous development. The first step is data acquisition, followed by data cleaning and data labeling. To use YOLOv5, bounding boxes have to be drawn around the anomalies, which is utilized through the *Make Sense* platform [84]. Once the whole data set is labeled, it is divided into training, testing, and validation sets as discussed in Section 3.5.

The workflow starts by initializing an empty git repository on Github, where DVC is installed and initialized. DVC creates a .dvc file that contains a unique md5 hash for linking the data set to the project. The next step is to add the code into the GitHub repository and training data to the initialized DVC repository. With the help of GitHub actions, automated triggers are generated for code and data validation as soon as these additions are made. For automated data validation, the full reference-based image quality assessment metric, SSIM, is utilized. One of the best quality images (according to the domain expert) is marked as a reference. The metric goes over the whole data set and evaluates a quality score for each image based on structural, contrast, and luminance information in comparison to the reference image. Every image which has a lower SSIM score than the average is removed from the data set. For code, basic unit and sanity checks are performed to check for any syntax-related errors.

In the next step, experiments are executed for finding the best-performing models using the Hyperopt library [88] while integrating the MLflow tracking in the training script. The MLflow tracking server can automatically track metrics, artifacts (models and plots), and parameters for these experiments. The runs are logged in an SQLite database, and artifacts are stored locally. These runs can be visualized with the help of the MLflow tracking user interface or can be accessed via the REST API. On top of this, the MLflow registry lets the user annotate models for model versioning. The best-performing model is packaged using the MLflow models as a python function.

6.2. Critical Discussion

Many organizations have mastered developing a model that works on a static dataset, but realizing real value from a model in the production environment is still a challenge. This challenge exists because the process of deploying the already developed model is not straightforward. In addition to the incremental complexity of the development environment, there is also dependency on the data. Information regarding the development environment, dataset, and hyperparameters is necessary for reproducing the same model. As the number of experiments increases, the amount of information required to be stored increases dramatically. All this information can not be logged manually, and there is a need for a tools stack that can ease the work of a data scientist and promote shareability between different data science teams. Muralidhar et al. show a real-life use case of MLOps in the financial analytics domain in [11]. They describe challenges and defective practices in terms of patterns (anti-patterns) that should be avoided for the successful deployment of ML models. The challenges faced by object detection models in real-world scenarios are well described by Ahmed et al. in [89]. The paper showcases that even the state of the art deep learning models suffer from performance degradation in challenging environments due to various factors. One solution for this is continual learning, where a model is updated frequently. If continual updates are required, then the development of newer models has to be automated. This is where MLOps showcases its potential.

MLOps has become a necessity for businesses to create actual practical workflows for computer vision applications. This is visible from the workflow defined in Figure 3. For example, suppose Git and DVC are not used for data and code versioning. In that case, the development team has to perform versioning by manually saving different folders with different datasets or outsourcing the complete data handling to a third party. Another complexity involved with images is the quality assurance of the data. Image quality assessment (IQA) is usually divided into two sections: Full Reference-based IQA (FR-IQA) and No Reference-based IQA (NR-IQA). The methods under NR-IQA require an additional learned model to quantify the quality of the images and have higher complexity. On the other hand, FR-IQA methods are less complex and have faster implementations. Some of the most common FR-IQA methods are presented in Table 1. For this workflow, the SSIM metric was chosen to perform automated quality assurance of the data due to the fact that it outperforms standard metrics such as MSE and PSNR. After data preprocessing and versioning, the following requirement is producing reproducible training experiments. This is achieved through MLflow's tracking functionality. In the above case, the YoloV5 large model is used, and the Hyperopt library optimizes hyperparameters. With very few tweaks, the other versions of YOLOv5 object detection models can also be used. This is also true if one decides to use any other optimization strategy such as genetic algorithms for hyperparameter optimization. The portrayed automated workflow helps in organizing this and other important information in a relational database. This information is vital for knowing what worked and ensuring reproducibility. The shown workflow gives the development team the freedom to try newer models and optimization strategies without changing other parts of the workflow. Through the MLflow's registry component, we can also save the different development stages and versions for a single model in the relational database. This information helps in debugging when a particular model fails in the deployment environment. Through the workflow portrayed in Figure 3, the models can be continuously improved, and this eases the complexity of the ML development process.

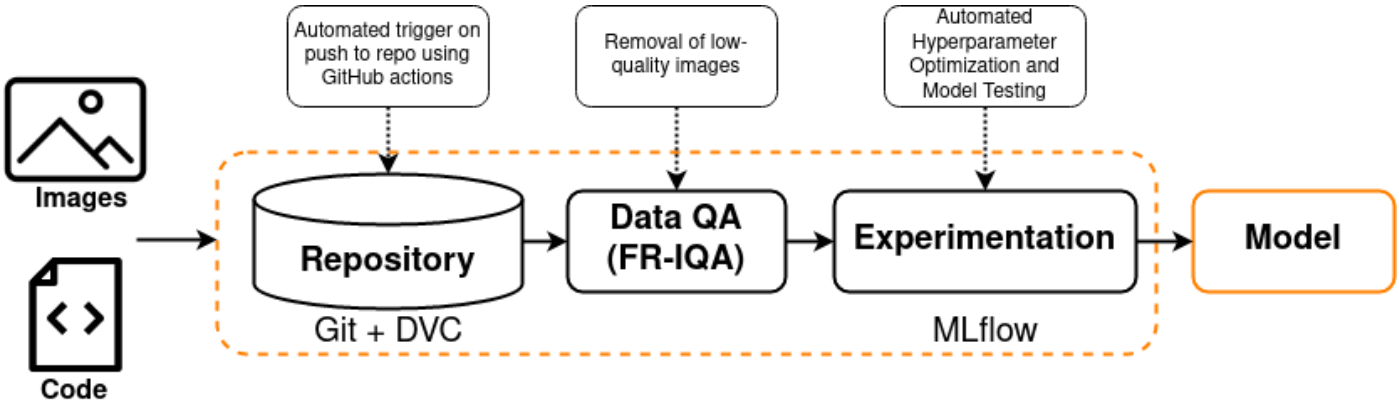

Figure 3. Basic MLOps workflow with automated training, Automl, model packaging, and model registry.

6.3. Fully Automating the MLOps Worklfow

In the workflow depicted in Figure 3, rolling back to the previous version of models is done manually. Furthermore, as the final product is a single model and not a pipeline, the model's continuous monitoring and continual learning are not performed. Furthermore, there are no options for model governance nor the implementation of a network layer or an orchestrator.

Additionally, a monitoring system is required to ensure a fast feedback loop for solving bugs and other problems that may be encountered in production. Some of the primary metrics for monitoring are listed in Tables 2–4. Once a fault is found, then an action policy is implemented. The action policy decides what changes should occur based on the alerts generated by the monitoring system. For example, when drift is detected, then the system has to calculate the ideal time of deploying a new model. All of these decisions are made based on the thresholds set by the domain expert, which should be highly related to the monitoring platform's alerting mechanism.

As an organization gets more comfortable with data science, more focus is given on containerization and full automation of ML workflows in addition to automation of CI/CD pipelines. This is also referred to as the final level of automation (MLOps Level 2) [33]. With this step, multiple pipelines can be deployed as a single entity rather than an individual model. These pipelines can be versioned. With containerization, container orchestrators like Kubernetes come into play. These all are important for managing the evolving workflow. For scalability and complete automation of MLOps workflow containers, orchestrators and resource managers form together an essential toolchain. With these tools, continuous and scalable ML pipelines are achieved.

7. Conclusions and Future Work

In this work, the potential of MLOps was presented. With the introduction of various actors and roles involved in MLOps workflows, an overview of responsibilities and supportive tools for the different phases were introduced. With the discussion of a real-world Deep Learning (DL)-based object detection use case, the benefits of managing such a project by interconnecting different concepts and supportive technologies were shown. By comparing various tools with respect to preprocessing, training, management, deployment, and operations in ML projects, a quick selection of problem-specific solutions or products is realized. This supports and simplifies the definition of the initial starting point for MLOps projects. Crucial metrics for monitoring ML systems are also listed (see Tables 2–4). These metrics are the key to obtaining a continual learning solution.

From the comparison of different tools depicted in Table 5, it can be seen that no single tool has the capability of realizing a fully automated MLOps workflow. Different tools have overlapping features, increasing redundancy. For example, DVC is not just a tool for data versioning; instead, it can also be used to track machine learning experiments, but due to no dashboard to visualize the experiments, MLflow is often chosen over DVC

for this task. Furthermore, there is also an expansion of the skills required by different actors in the ML lifecycle to create any real business value. For example, earlier, the role of data scientist was limited to experimentation and modeling. However, it can be seen from Figure 2 that close collaboration with operation or software engineers is needed for the successful deployment of an ML model.

In future work, the potentials of AutoML capabilities for MLOps workflows and their deep integration and orchestration within such operations could be investigated. By revealing the combinations of state-of-the-art software, their infrastructural restrictions and requirements could be focused on. As the tool variety for supporting the different phases in ML projects is constantly evolving, many operational tools are expected to be refined in the future. Exciting and novel offerings for interconnection capabilities with other tools and a more holistic metric support may be worth further investigation. The implementation of expert interviews regarding their experience with interconnecting and integrating various tools for creating complete MLOps pipelines and ecosystems are of interest, too. As concepts for assuring data quality constantly emerge, pursuing their integration into MLOps tools and workflows appears to be worthwhile.

Author Contributions: Conceptualization, P.R. and M.M.; methodology, P.R. and M.M.; formal analysis, P.R. and M.M.; investigation, P.R. and M.M.; resources, P.R. and M.M..; data curation, P.R. and M.M.; writing—original draft preparation, P.R. and M.M.; writing—review and editing, P.R., M.M., C.R. and D.O.-A.; supervision, C.R. and D.O.-A.; project administration, C.R.; funding acquisition, C.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by by the Ministry of Science, Research and the Arts of the State of Baden-Württemberg (MWK BW) under reference number 32-7547.223-6/12/4.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: The contents of this publication are taken from the research project "(Q-AMeLiA)—Quality Assurance of Machine Learning Applications", which is supervised by Hochschule Furtwangen University (Christoph Reich, IDACUS).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.F.; Dennison, D. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Cambridge, MA, USA, 2015; Volume 28.
2. Goyal, A. Machine Learning Operations. In *International Journal of Information Technology Insights & Transformations [ISSN: 2581-5172 (Online)]*; Eureka Journals: Pune, India 2020; Volume 4.
3. Raj, E.; Westerlund, M.; Espinosa-Leal, L. Reliable Fleet Analytics for Edge IoT Solutions. *arXiv* **2021**, arXiv:2101.04414.
4. Rai, R.K. Intricacies of unstructured data. *EAI Endorsed Trans. Scalable Inf. Syst.* **2017**, doi:10.4108/eai.25-9-2017.153151, *4*.
5. Mohammadi, B.; Fathy, M.; Sabokrou, M. Image/Video Deep Anomaly Detection: A Survey. *arXiv* **2021**, arXiv:2103.01739.
6. Shrivastava, S.; Patel, D.; Zhou, N.; Iyengar, A.; Bhamidipaty, A. DQLearn: A Toolkit for Structured Data Quality Learning. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; pp. 1644–1653.
7. Tamburri, D.A. Sustainable MLOps: Trends and Challenges. In Proceedings of the 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 1–4 September 2020; pp. 17–23.
8. Fursin, G.; Guillou, H.; Essayan, N. CodeReef: an open platform for portable MLOps, reusable automation actions and reproducible benchmarking. *arXiv* **2020**, arXiv:2001.07935.
9. Granlund, T.; Kopponen, A.; Stirbu, V.; Myllyaho, L.; Mikkonen, T. MLOps Challenges in Multi-Organization Setup: Experiences from Two Real-World Cases. *arXiv* **2021**, arXiv:2103.08937.
10. Zhao, Y. Machine Learning in Production: A Literature Review. 2021. Available online: https://staff.fnwi.uva.nl/a.s.z.belloum/LiteratureStudies/Reports/2021-LiteratureStudy-report-Yizhen.pdf (accessed on 27 July 2021).
11. Muralidhar, N.; Muthiah, S.; Butler, P.; Jain, M.; Yu, Y.; Burne, K.; Li, W.; Jones, D.; Arunachalam, P.; McCormick, H.S.; et al. Using AntiPatterns to avoid MLOps Mistakes. *arXiv* **2021**, arXiv:2107.00079.

12. Silva, L.C.; Zagatti, F.R.; Sette, B.S.; dos Santos Silva, L.N.; Lucrédio, D.; Silva, D.F.; de Medeiros Caseli, H. Benchmarking Machine Learning Solutions in Production. In Proceedings of the 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 14–17 December 2020; pp. 626–633.
13. Sureddy, M.R.; Yallamula, P. A Framework for Monitoring Data Warehousing Applications. *Int. Res. J. Eng. Technol.* **2020**, *7*, 7023–7029.
14. Shivakumar, S.K., Web Performance Monitoring and Infrastructure Planning. In *Modern Web Performance Optimization: Methods, Tools, and Patterns to Speed Up Digital Platforms*; Apress: Berkeley, CA, USA, 2020; pp. 175–212. doi:10.1007/978-1-4842-6528-4_7.
15. Sebastian-Coleman, L. Chapter 4—Data Quality and Measurement. In *Measuring Data Quality for Ongoing Improvement*; Sebastian-Coleman, L., Ed.; MK Series on Business Intelligence; Morgan Kaufmann: Boston, MA, USA, 2013; pp. 39–53. doi:10.1016/B978-0-12-397033-6.00004-3.
16. Schelter, S.; Lange, D.; Schmidt, P.; Celikel, M.; Biessmann, F.; Grafberger, A. Automating large-scale data quality verification. *Proc. VLDB Endow.* **2018**, *11*, 1781–1794.
17. Taleb, I.; Serhani, M.A.; Dssouli, R. Big data quality: A survey. In Proceedings of the 2018 IEEE International Congress on Big Data (BigData Congress), San Francisco, CA, USA, 2–7 July 2018; pp. 166–173.
18. Barrak, A.; Eghan, E.E.; Adams, B. On the Co-evolution of ML Pipelines and Source Code - Empirical Study of DVC Projects. In Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Honolulu, HI, USA, 9–12 March 2021; pp. 422–433. doi:10.1109/SANER50967.2021.00046.
19. Ramasubramanian, K.; Singh, A. Machine learning model evaluation. In *Machine Learning Using R*; Apress: Berkeley, CA, USA; 2017; pp. 425–464. doi: 10.1007/978-1-4842-2334-5_7.
20. Mehmood, H.; Kostakos, P.; Cortes, M.; Anagnostopoulos, T.; Pirttikangas, S.; Gilman, E. Concept drift adaptation techniques in distributed environment for real-world data streams. *Smart Cities* **2021**, *4*, 349–371.
21. Hutter, F.; Kotthoff, L.; Vanschoren, J. (Eds.) *Automated Machine Learning: Methods, Systems, Challenges*; Springer: Berlin, Germany, 2018; In Press. Available online: http://automl.org/book (accessed on 27 July 2021).
22. Zöller, M.A.; Huber, M.F. Survey on automated machine learning. *arXiv* **2019**, arXiv:1904.12054.
23. Peng, G.; Lacagnina, C.; Downs, R.R.; Ramapriyan, H.; Ivánová, I.; Ganske, A.; Jones, D.; Bastin, L.; Wyborn, L.; Bastrakova, I.; et al. International Community Guidelines for Sharing and Reusing Quality Information of Individual Earth Science Datasets. OSF Preprints, 16 April 2021. Available online: https://osf.io/xsu4p (accessed on 27 August 2021).
24. Raj, E. *Engineering MLOps: Rapidly Build, Test, and Manage Production-Ready Machine Learning Life Cycles at Scale*; Packt Publishing: Birmingham, UK 2021.
25. Wang, D.; Liao, Q.V.; Zhang, Y.; Khurana, U.; Samulowitz, H.; Park, S.; Muller, M.; Amini, L. How Much Automation Does a Data Scientist Want? *arXiv* **2021**, arXiv:2101.03970.
26. AWS MLOps Framework. Available online: https://aws.amazon.com/solutions/implementations/aws-mlops-framework/ (accessed on 14 September 2021).
27. Sharma, S. *The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise*; John Wiley & Sons: Hoboken, NJ, USA, 2017.
28. Karamitsos, I.; Albarhami, S.; Apostolopoulos, C. Applying DevOps practices of continuous automation for machine learning. *Information* **2020**, *11*, 363.
29. Mäkinen, S.; Skogström, H.; Laaksonen, E.; Mikkonen, T. Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? *arXiv* **2021**, arXiv:2103.08942.
30. Treveil, M.; Omont, N.; Stenac, C.; Lefevre, K.; Phan, D.; Zentici, J.; Lavoillotte, A.; Miyazaki, M.; Heidmann, L. *Introducing MLOps*; O'Reilly Media: Sebastopol, CA, USA, 2020.
31. Lu, J.; Liu, A.; Dong, F.; Gu, F.; Gama, J.; Zhang, G. Learning under concept drift: A review. *IEEE Trans. Knowl. Data Eng.* **2018**, *31*, 2346–2363.
32. Baylor, D.; Haas, K.; Katsiapis, K.; Leong, S.; Liu, R.; Menwald, C.; Miao, H.; Polyzotis, N.; Trott, M.; Zinkevich, M. Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform. In Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML 19), Santa Clara, CA, USA, 20 May 2019; pp. 51–53.
33. Google. MLOps: Continuous Delivery and Automation Pipelines in Machine Learning. Available online: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning/ (accessed on 3 May 2021).
34. Maydanchik, A. *Data Quality Assessment*; Technics Publications: Basking Ridge, NJ, USA, 2007.
35. Verheul, I.; Imming, M.; Ringerma, J.; Mordant, A.; Ploeg, J.L.V.D.; Pronk, M. Data Stewardship on the map: A study of tasks and roles in Dutch research institutes. 2019. Available online: https://zenodo.org/record/2669150#.YUw2BH0RVPY (accessed on 27 August 2021).
36. Wende, K. A model for data governance–Organising accountabilities for data quality management. In Proceedings of the Data Stewardship on the Map: A Study of Tasks and Roles in Dutch Research Institutes, Toowoomba, Australia, 5–7 December 2007.
37. Pergl, R.; Hooft, R.; Suchánek, M.; Knaisl, V.; Slifka, J. "Data Stewardship Wizard": A tool bringing together researchers, data stewards, and data experts around data management planning. *Data Sci. J.* **2019**, *18*, 59.
38. Peng, G.; Ritchey, N.A.; Casey, K.S.; Kearns, E.J.; Privette, J.A.; Saunders, D.; Jones, P.; Maycock, T.; Ansari, S. Scientific Stewardship in the Open Data and Big Data Era-Roles and Responsibilities of Stewards and Other Major Product Stakeholders. 2016. Available online: https://www.dlib.org/dlib/may16/peng/05peng.html (accessed on 27 August 2021).

39. Mons, B. *Data Stewardship for Open Science: Implementing FAIR Principles*; CRC Press: Boca Raton, FL, USA, 2018.
40. Mons, B.; Neylon, C.; Velterop, J.; Dumontier, M.; da Silva Santos, L.O.B.; Wilkinson, M.D. Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud. *Inf. Serv. Use* **2017**, *37*, 49–56.
41. Zubair, N.; Hebbar, K.; Simmhan, Y. Characterizing IoT data and its quality for use. *arXiv* **2019**, arXiv:1906.10497.
42. Gudivada, V.; Apon, A.; Ding, J. Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *Int. J. Adv. Softw.* **2017**, *10*, 1–20.
43. Dong, X.L.; Rekatsinas, T. Data integration and machine learning: A natural synergy. In Proceedings of the 2018 International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018; pp. 1645–1650.
44. Kißkalt, D.; Mayr, A.; Lutz, B.; Rögele, A.; Franke, J. Streamlining the development of data-driven industrial applications by automated machine learning. *Procedia CIRP* **2020**, *93*, 401–406.
45. Lee, Y.; Scolari, A.; Chun, B.G.; Weimer, M.; Interlandi, M. From the Edge to the Cloud: Model Serving in ML. NET. *IEEE Data Eng. Bull.* **2018**, *41*, 46–53.
46. Wang, Z.; Bovik, A.C.; Sheikh, H.R.; Simoncelli, E.P. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612.
47. Li, C.; Bovik, A.C. Content-partitioned structural similarity index for image quality assessment. *Signal Process. Image Commun.* **2010**, *25*, 517–526.
48. Wang, Z.; Li, Q. Information content weighting for perceptual image quality assessment. *IEEE Trans. Image Process.* **2010**, *20*, 1185–1198.
49. Li, C.; Bovik, A.C. Three-component weighted structural similarity index. In Proceedings of the Image Quality and System Performance VI. International Society for Optics and Photonics, 19–21 January 2009, San Jose, CA, USA; Volume 7242, p. 72420Q.
50. de Freitas Zampolo, R.; Seara, R. A comparison of image quality metric performances under practical conditions. In Proceedings of the IEEE International Conference on Image Processing 2005, Genoa, Italy, 11–14 September 2005; Volume 3, pp. 1192–1195. doi:10.1109/ICIP.2005.1530611.
51. Sheikh, H.R.; Bovik, A.C. Image information and visual quality. *IEEE Trans. Image Process.* **2006**, *15*, 430–444.
52. Zhang, L.; Zhang, L.; Mou, X.; Zhang, D. FSIM: A feature similarity index for image quality assessment. *IEEE Trans. Image Process.* **2011**, *20*, 2378–2386.
53. Xue, W.; Zhang, L.; Mou, X.; Bovik, A.C. Gradient magnitude similarity deviation: A highly efficient perceptual image quality index. *IEEE Trans. Image Process.* **2013**, *23*, 684–695.
54. Egiazarian, K.; Astola, J.; Ponomarenko, N.; Lukin, V.; Battisti, F.; Carli, M. New full-reference quality metrics based on HVS. In Proceedings of the Second International Workshop on Video Processing and Quality Metrics, Scottsdale, AZ, USA, 22–24 January 2006; Volume 4.
55. Larson, E.C.; Chandler, D.M. Most apparent distortion: full-reference image quality assessment and the role of strategy. *J. Electron. Imaging* **2010**, *19*, 011006.
56. Lee, D.; Plataniotis, K.N. Towards a full-reference quality assessment for color images using directional statistics. *IEEE Trans. Image Process.* **2015**, *24*, 3950–3965.
57. Zhang, L.; Shen, Y.; Li, H. VSI: A visual saliency-induced index for perceptual image quality assessment. *IEEE Trans. Image Process.* **2014**, *23*, 4270–4281.
58. Mattson, P.; Cheng, C.; Coleman, C.; Diamos, G.; Micikevicius, P.; Patterson, D.; Tang, H.; Wei, G.Y.; Bailis, P.; Bittorf, V.; et al. Mlperf training benchmark. *arXiv* **2019**, arXiv:1910.01500.
59. MLflow. MLflow. Available online: https://mlflow.org/ (accessed on 27 July 2021).
60. Polyaxon—Machine Learning at Scale. Available online: https://polyaxon.com/ (accessed on 27 July 2021).
61. Kedro: A Python Framework for Creating Reproducible, Maintainable and Modular Data Science Code. Available online: https://github.com/quantumblacklabs/kedro (accessed on 27 July 2021).
62. Baylor, D.; Breck, E.; Cheng, H.T.; Fiedel, N.; Foo, C.Y.; Haque, Z.; Haykal, S.; Ispir, M.; Jain, V.; Koc, L.; et al Tfx: A tensorflow-based production-scale machine learning platform. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 13–17 August 2017; pp. 1387–1395.
63. ZenML. Available online: https://zenml.io/ (accessed on 27 July 2021).
64. H2O: Tfully Open Source, Distributed in-Memory Machine Learning Platform. Available online: https://www.h2o.ai/products/h2o/ (accessed on 2 September 2021).
65. Kubeflow: The Machine Learning Toolkit for Kubernetes. Available online: https://www.kubeflow.org/ (accessed on 27 July 2021).
66. Flyte: The Workflow Automation Platform for Complex, Mission-Critical Data and ML Processes at Scale. Available online: https://flyte.org/ (accessed on 27 July 2021).
67. Apache Airflow, a Platform Created by the Community to Programmatically Author, Schedule and Monitor Workflows. Available online: https://airflow.apache.org/ (accessed on 27 July 2021).
68. DVC: Open-Source Version Control System for Machine Learning Projects. Available online: https://dvc.org/ (accessed on 27 July 2021).
69. The Data Foundation for Machine Learning. Available online: https://www.pachyderm.com/ (accessed on 27 July 2021).
70. Quilt. Available online: https://quiltdata.com/ (accessed on 27 July 2021).

71. Great Expectations. Available online: https://greatexpectations.io/ (accessed on 27 July 2021).
72. Git Large File Storage (LFS). Available online: https://git-lfs.github.com/ (accessed on 27 July 2021).
73. Continuous Machine Learning (CML). Available online: https://cml.dev/ (accessed on 27 July 2021).
74. GitHub Actions. Available online: https://github.com/features/actions (accessed on 27 July 2021).
75. circleci. Available online: https://circleci.com/ (accessed on 27 July 2021).
76. gocd. Available online: https://www.gocd.org/ (accessed on 27 July 2021).
77. Cortex. Available online: https://www.cortex.dev/ (accessed on 27 July 2021).
78. Seldon Core. Available online: https://github.com/SeldonIO/seldon-core (accessed on 27 July 2021).
79. BentoML. Available online: https://github.com/bentoml/BentoML (accessed on 27 July 2021).
80. Prometheus—Monitoring System and Time Series Database. Available online: https://prometheus.io/ (accessed on 27 July 2021).
81. Kibana. Available online: https://www.elastic.co/kibana/ (accessed on 27 July 2021).
82. Grafana: The Open Observability Platform. Available online: https://grafana.com (accessed on 27 July 2021).
83. Lable Studio. Available online: https://labelstud.io/ (accessed on 27 July 2021).
84. Make Sense. Available online: https://www.makesense.ai/ (accessed on 27 July 2021).
85. Tao, X.; Zhang, D.; Ma, W.; Liu, X.; Xu, D. Automatic metallic surface defect detection and recognition with convolutional neural networks. *Appl. Sci.* **2018**, *8*, 1575.
86. Ruff, L.; Kauffmann, J.R.; Vandermeulen, R.A.; Montavon, G.; Samek, W.; Kloft, M.; Dietterich, T.G.; Müller, K.R. A unifying review of deep and shallow anomaly detection. In Proceedings of the IEEE, Xiamen, China, 28–30 July 2021. doi:10.1109/JPROC.2021.3052449.
87. Ultralytics. YOLOv5. Available online: https://github.com/ultralytics/yolov5 (accessed on 22 July 2021).
88. Bergstra, J.; Yamins, D.; Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 115–123.
89. Ahmed, M.; Hashmi, K.A.; Pagani, A.; Liwicki, M.; Stricker, D.; Afzal, M.Z. Survey and Performance Analysis of Deep Learning Based Object Detection in Challenging Environments. *Sensors* **2021**, *21*, 5116.