applied sciences

MDPI

Article

From DevOps to MLOps: Overview and Application to Electricity Market Forecasting

Rakshith Subramanya [1,*], Seppo Sierla [1] and Valeriy Vyatkin [1,2]

[1] Department of Electrical Engineering and Automation, School of Electrical Engineering, Aalto University, 02150 Espoo, Finland
[2] Department of Computer Science, Electrical and Space Engineering, Luleå Tekniska Universitet, 97187 Luleå, Sweden
* Correspondence: rakshith.subramanya@aalto.fi

Abstract: In the Software Development Life Cycle (SDLC), Development and Operations (DevOps) has been proven to deliver reliable, scalable software within a shorter time. Due to the explosion of Machine Learning (ML) applications, the term Machine Learning Operations (MLOps) has gained significant interest among ML practitioners. This paper explains the DevOps and MLOps processes relevant to the implementation of MLOps. The contribution of this paper towards the MLOps framework is threefold: First, we review the state of the art in MLOps by analyzing the related work in MLOps. Second, we present an overview of the leading DevOps principles relevant to MLOps. Third, we derive an MLOps framework from the MLOps theory and apply it to a time-series forecasting application in the hourly day-ahead electricity market. The paper concludes with how MLOps could be generalized and applied to two more use cases with minor changes.

Keywords: continuous software engineering; DevOps; electricity market; Machine Learning; MLOps; time-series analysis

Citation: Subramanya, R.; Sierla, S.; Vyatkin, V. From DevOps to MLOps: Overview and Application to Electricity Market Forecasting. *Appl. Sci.* **2022**, *12*, 9851. https://doi.org/10.3390/app12199851

Academic Editor: Emanuele Carpanzano

Received: 26 August 2022
Accepted: 27 September 2022
Published: 30 September 2022

1. Introduction

Sufficient motivation for the DevOps process emerged around 2009 [1]. At this time, Development and Operations teams struggled to achieve smooth rollouts of software products. The main reason for this struggle was that the software developers were not concerned about deployments and the operation teams were not concerned about the development processes. DevOps is a set of processes that utilizes cross-functional teams to build, test, and release software faster, in a reliable and repeatable manner, through automation [1–3]. Recently, investment in Machine Learning (ML) applications has enabled stakeholders to solve complex business use cases that were difficult to solve. However, in most cases, ML applications are only a tiny part of a more extensive software system, and this small fraction of ML code is surrounded by a variety of software, libraries, and configuration files [4]. Hence, the main challenge in ML applications is to build continuous software engineering practices [5,6], such as DevOps [7], which can promise stakeholders the seamless integration and deployment known as MLOps [8,9]. MLOps refers to DevOps principles applied to ML applications. This paper introduces both DevOps and MLOps, provides a detailed explanation of both, and explains how to implement MLOps from the perspective of DevOps. Before diving deep into these technologies, it is helpful to understand some history behind DevOps and how MLOps has evolved from DevOps. This paper makes three contributions:

1. The literature on the motivations and the state of the art of MLOps is reviewed.
2. An overview of MLOps theory and DevOps theory relevant to the implementation of MLOps is presented, and an MLOps framework is proposed.
3. The proposed framework is applied to a time-series forecasting application as a case study. The case study is implemented with MLOps pipelines.

Most importantly, this paper systematically presents the concept of MLOps from DevOps and explores how to implement and extend the generic MLOps pipeline to multiple use cases. These two aspects are the motivation and significance of this paper. The remainder of the article is organized as follows: Section 2 reviews related works from the context of MLOps. Section 3 provides an overview of DevOps principles that lead to the development of MLOps, along with our generic MLOps framework. Section 4 presents the application of the proposed generic MLOps use case for forecasting an hourly day-ahead electricity market price.

2. Related Work

2.1. Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a methodology with defined processes for creating high-quality software [10]. SDLC processes include different phases, such as planning, analysis, design, and implementation. The Waterfall model, Spiral model, Iterative model, Prototype model, V-model, Rapid Development model (RAD), and Agile model are some of the major SDLC models [10,11]. For successful project implementation, it is crucial to select a proper SDLC model depending on different parameters, such as software complexity and type [12,13]. Dayal Chauhan et al. [14] analyzed the impacts of various SDLC methods on the cost and risk of projects. Several authors have classified all of the available SDLC models into two types of methodology: heavyweight and lightweight methodologies [10,11,15]. Heavyweight methodologies are mainly process-oriented, might not entertain requirement changes, and emphasize documentation. The Waterfall, Spiral, and Incremental models are a few examples of heavyweight methodologies [15,16]. Lightweight methodologies are mainly people-oriented, entertain frequent requirement changes, have short development life cycles, and involve the customer. The Prototyping, RAD, and Agile models are a few examples of lightweight methodologies [15,16].

Several authors have compared lightweight and heavyweight methodologies and provided more insights on the SDLC selection process. For instance, Ben-Zahia and Jaluta [11] discussed the criteria for selecting proper SDLC models based on people, process, or plan orientations. A few authors have defined a third methodology called the hybrid development methodology, which uses both heavyweight and lightweight methods [17]. Khan et al. [10] used the analytic hierarchy process to select the best SDLC model from all three methodologies. Among the lightweight, heavyweight, and hybrid SDLC methodologies, Waterfall and Agile are the most used SDLC methods, based on different parameters such as usability [18,19], cost [20,21], safety [22,23], and customer involvement [24]. Several authors have shown that customers are transitioning to Agile from the traditional Waterfall SDLC due to the advantages of Agile, including short development life cycle, frequent changes, customer involvement, and usability [19,22]. However, Agile SDLC methods have also been criticized by some practitioners in cases where the requirements do not often change [25] or where there are human-related challenges related to a lack of emotional intelligence. Various authors [26,27] have analyzed issues such as quality and productivity with traditional software development strategies such as Waterfall and compared them to the Agile methodology, where clear advantages can be seen in Agile.

2.2. Agile and DevOps

Agile methodologies are the most widely implemented project management approaches in modern software systems. The 12 principles of the Agile Manifesto [28,29] characterize the process integrity and methods of Agile project management, which are applied to different Agile methodologies. Scrum, extreme programming, lean software development, and crystal methodologies are some of the Agile methodologies [14,30,31]. Ever-changing business needs demand a continuous process in software development, delivery, and system operations [32]. Implementing these continuous software practices in Agile has enabled fast delivery of software [33]. Martin Fowler introduced the idea of Continuous Integration (CI) and, later, J. Humble and D. Farley extended these ideas into

Continuous Delivery (CD) as a concept of the deployment pipeline [34]. Several authors, including Arachchi et al. [34] and Süß et al. [35], conducted research on automating CI and CD for Agile. With Agile, faster software development, quality improvement, frequent requirement changes, and customer involvement are achieved throughout the project. Nevertheless, the structural gap between the Development and Operations teams remains with the Agile methodologies. Development and Operations (DevOps) practices close this gap [36].

The working nature of software developers and operational professionals is different, and they work in isolation. This isolation might cause some conflicts between them [3]. The Development and Operations teams work under different departments and leadership [37]. Working in isolation leads to different Key Performance Indices (KPIs) [38], which are important in most organizations as they are used to define the performance of an individual or a team [39]. Not only do KPIs define the performance alone, they are also a function of several metrics [40]. An example of a developer KPI could be the time needed to roll out a feature with minor or no bugs. For the operations team, one KPI could be the time taken to roll out a feature with less or no downtime. To improve the KPIs, each team should concentrate on the task at hand. As the size of the software grows, maintainability and scalability become increasingly serious concerns [41,42]. For instance:

- An error might occur when the operations team tries to deploy a new feature to the production environment [43,44]. However, the same code might work on the developer's machine and the Quality Assurance (QA) instance.
- The new codebase might break old features.
- It is difficult to track what has changed.
- The software might not serve its core purpose at all.

These are only a subset of the issues that can arise due to the Development and Operations teams working in isolation and chasing their own KPIs. None of the product owners [45], developers [46], or team leaders from either team are responsible for addressing the issues mentioned above. Even though these issues can be solved, this requires more discussions and code exchanges between the Development and Operations teams. Various studies [47–49] have been performed to understand the effects of such impacts. Any software project has one more key stakeholder—the customer [50,51]. The customer plays a significant role in the software life cycle [13], as the customer's tolerance to risk and ability to support collaborative routines influences key parameters such as project cost and implementation time.

These issues are the driving factors that led to the formulation of the Agile and DevOps methodologies [52–54]. Agile focuses on formal requirement gathering, on small–medium but rapid releases and, finally, on continuous customer feedback. It enforces collaboration between various teams to ensure rapid reaction times to ever-changing consumer needs [55,56]. This collaboration contrasts with the traditional project management approach, which concentrates on long timelines and schedules. DevOps is the extension of the Agile methodology [57,58], and the two can work in tandem [37,51], as shown in Figure 1.

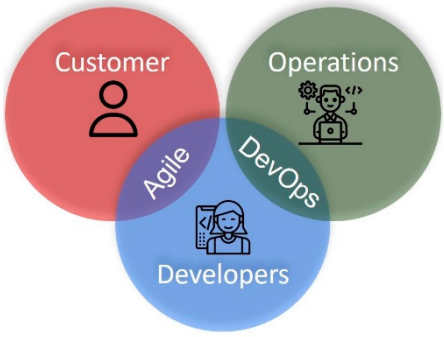

Figure 1. Agile and DevOps.

The Agile workflow assumes that the software development process is divided into multiple sprints [59,60] and the customer is notified about the changes. A sprint is the smallest duration of Agile, where a team works on an assigned task [60]. The main goal for the Development and Operations teams in a sprint is to produce a stable software release in every sprint cycle [61]. Combining Agile activities with the DevOps framework is reasonable considering the software development and delivery aspects. One such example is integrating Agile tools with DevOps tools.

2.3. MLOps from DevOps

With the successful adoption of DevOps, various organizations are trying to incorporate continuous practices in ML system development [62]. ML developers and operations teams are trying to adopt DevOps concepts to ML systems for end-to-end life-cycle automation. Containerized microservices and cloud-based DevOps have seen good stability and a reasonable success rate in production deployments [63]. For example, Kubeflow [64,65] is an ML toolkit for Kubernetes, available for almost all major cloud providers. Such toolkits aim to create ML workflows in which containers are already present. Karamitsos et al. [66] discussed such concepts and issues of applying DevOps practices to ML applications, while John et al. [62] discusses DevOps applications in ML systems. Data play an influential role in any ML application. Unlike conventional software development practices, the SDLC of an ML application revolves around data. In most cases, the core ML code is minimal, but it must integrate with some significant components of a bigger system. Sculley et al. [4] analyzed such systems and explored the hidden technical debt. The complexity increases when ML applications are deployed on the cloud or interfaced with web APIs. Banerjee et al. [67] proposed operationalizing ML applications for such an environment, and MLOps could be implemented for such hybrid cloud deployments.

Several authors have investigated MLOps. Makinen et al. [9] studied the state of ML to understand the extent of MLOps required and to analyze the issues with data; however, the authors did not consider deployment. Some research has been performed on the trends and challenges in MLOps. Tamburri et al. [68] recapped trends in terms of properties such as the fairness and accountability of sustainable ML operations of a software system. Several works propose the MLOps framework for applications [62,69] such as the automotive industry, supply chain management, and IoT. Granlund et al. [70] presented issues related to the MLOps pipeline when multiple organizations are involved, highlighting factors such as scalability. Different cloud providers offer MLOps solutions as a service so that the whole ML product life cycle can be managed on the cloud. Azure MLOps [71] and AWS SageMaker for MLOps [72] are some examples of such cloud services. A few cloud providers have published a detailed guide on the MLOps life cycle for the purpose of building ML applications and MLOps pipelines on the cloud or on-premises servers [73].

2.4. Summary of MLOps from DevOps

This section presents state-of-the-art MLOps articles mentioned in Section 2.3 in terms of methodology, novelty, and results. Table 1 provides the summary.

Table 1. Summary of state-of-the-art MLOps articles.

Paper	Methodology	Novelty	Result
[62]	Systematic literature review along with a grey literature review to derive a framework	MLOps framework that describes the activities involved in the continuous development of the ML model	Framework validation in three embedded systems case companies
[65]	Verified the feasibility of creating an ML pipeline with CI/CD capabilities on various appliances with specific hardware configuration	Performance evaluation of ML pipeline platforms characterized by Kubeflow on different models according to various metrics.	Consumption of time and resources concerning the ML platform and computational models.

Table 1. *Cont.*

Paper	Methodology	Novelty	Result
[66]	Review of the two DevOps components CI and CD, in the ML context	ML manual and automated pipeline design with CI/CD components	Scaling ML models
[67]	Five issues in ML-based solutions for performance diagnostics	MLOps pipeline to resolve the five challenges in the performance diagnostics	Fully automated pipeline for continuously training the new models upon the arrival of new data
[9]	Surveyed and compiled responses from ML professionals to investigate the role of MLOps in their daily activities	Survey with questions and goals	Based on survey, presented data on challenges based on survey data, ML problem types, and future plans with ML
[68]	A brief overview of the state-of-the-art MLOps and an overview of the organizational and educational structures around AI software operations	Challenges and trends in AI software operations	Challenges in educating AI operations and properties to be supported by software for the domain it was designed for
[69]	Multiple IoT experimental setups. Each setup is equipped with an IoT device and an edge device	An automated framework for MLOps at the edge, i.e., edge MLOps	Deployed and monitored IoT data and edge node operations in real time
[70]	Study integration between two organizations in detail for a multi-organization setup	Addressed scaling of ML to a multi-organization context	Integrated challenges of the ML pipeline, datasets, models, and monitoring.

3. An Overview of DevOps and MLOps

3.1. DevOps

DevOps is a set of practices or fault-tolerant workflows built to increase software quality by providing continuous development and delivery through end-to-end automation [35,53,74]. DevOps practices bring together the development, testing, and operational software development teams through automation [2]. DevOps enables a shorter code–build–deploy loop with a better end product [4,75]. A typical DevOps workflow is shown in Figure 2.

3.1.1. DevOps Workflow and Components

The main focus of DevOps is to automate the software delivery process throughout, thereby ensuring continuous delivery and the feedback loop of the software [76]. Continuous delivery combines the development, testing, and deployment processes into one streamlined operation. The primary goal is to quicken the whole process through automation. If both Development and Operations teams are practicing DevOps, they can quickly deploy code improvements, improving the transparency between two teams and allowing the end user to see changes quickly [53,74,77]. Due to continuous deployment and the involvement of customers in the DevOps workflow, the customers do not have to wait for a monthly/quarterly/yearly software release cycle [53,78] to test or provide feedback about the software.

The DevOps workflow shown in Figure 2 helps the teams to build, test, and deploy software quickly and efficiently through a combination of tools and practices, from development to maintenance. DevOps reduces Time to Market (TTM) and enables Agile software development processes [79,80]. These DevOps components are closely related to Agile. DevOps is the next step in the evolution of Agile methodologies [53,79]. The following subsections discuss the components—or phases [53]—within the DevOps workflow [58,79,81].

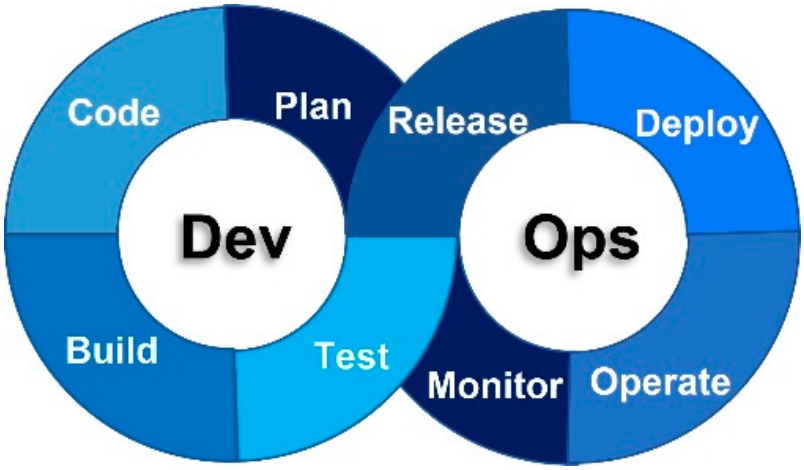

Figure 2. Typical DevOps workflow adapted from [66,79].

Plan

In this phase, requirements are defined, and the initial execution plan is created. The tools used for DevOps help the user and the developers to be constantly in sync. In this phase, if Agile is used as the SDLC, the user stories and tech stories should be defined with the teams [82]. Various issue-tracking and project-management tools, such as Jira [83], are used in this phase to help with the planning.

Code

Coding is the first step in DevOps automation, and all of the team members should adhere to the agreed coding standards and best practices [2,84]. Test-Driven Development (TDD), Acceptance Test-Driven Development (ATDD), and Behavior-Driven Development (BDD) are a few of the best practices [84]. One of the significant areas in coding that is still highly neglected is versioning of the software via source control. Versioning not only helps to maintain the software, but also helps to automate the DevOps workflow properly [85,86]. It is essential to implement good practices within the source control, such as pull requests, proper branching, and commit messages [87–89]. Several authors [90,91] have shown that artifact traceability can be achieved if performed adequately with good practices.

A branching strategy is used to share and collaborate the code changes among development teams. Feature-branching-based versioning and trunk-based versioning are the two most widely used branching strategies. Some versioning tools include Git, Subversion (SVN), and Team Foundation Server (TFS). Figure 3 shows a feature branching strategy with different types of branches [92,93], while Figure 4 shows typical trunk-based versioning [94,95].

Build

The software build usually refers to the whole software package, consisting of the business logic, the software dependencies, and the environment [96]. A few authors refer to the build phase as the verify phase [81]. In either case, the main aim of DevOps build systems is to evaluate the correctness of software artifacts [81,97]. Typically, three standard deployment instances are used while developing an application—development, Quality Assurance (QA), and production instances [98,99]. The build systems in DevOps make sure that code integrity is maintained. Software build systems are highly dependent on software configuration management systems, and multiple builds are possible among these environments [97,100], including private system builds. Several authors, including Leite et al. [101], mention various build tools for DevOps.

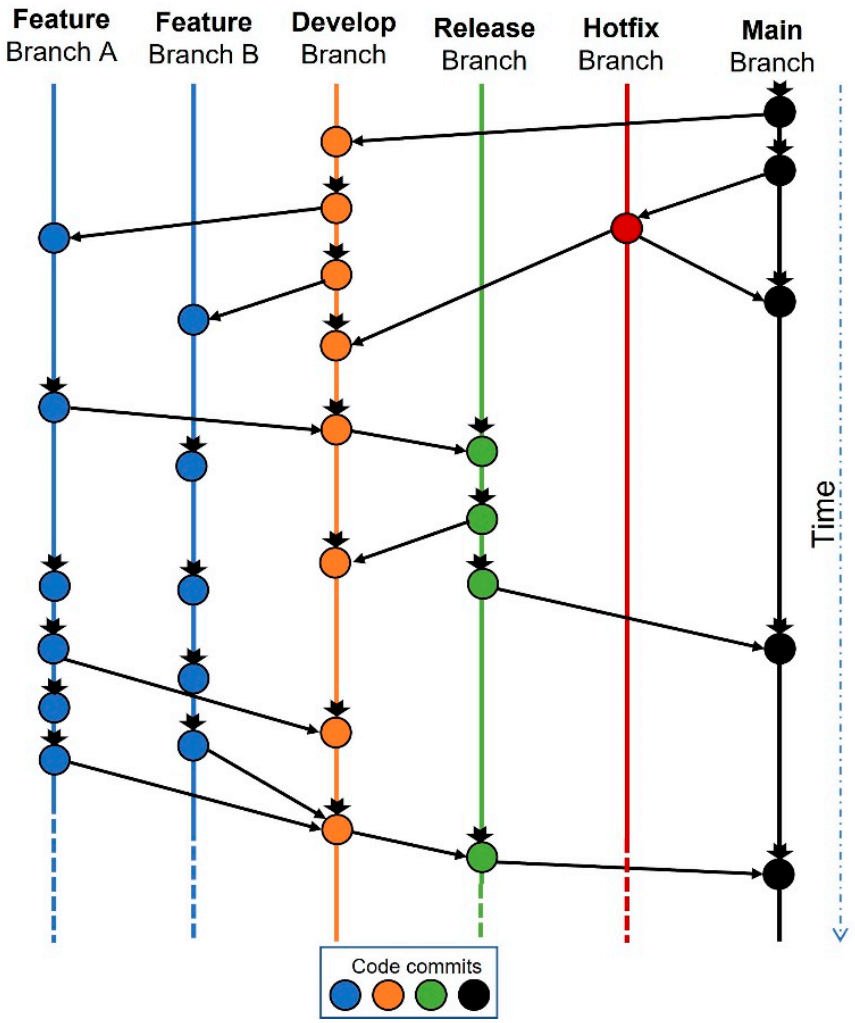

Figure 3. Feature branching strategy adapted from [92,93].

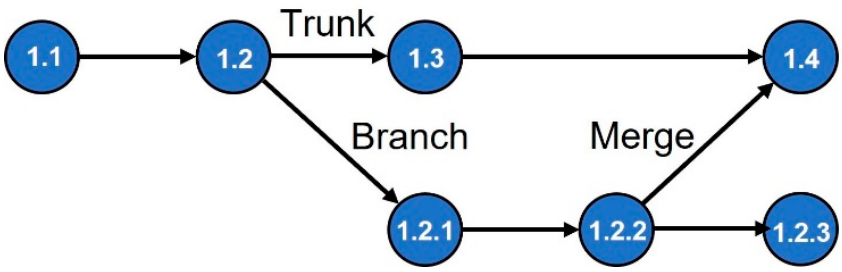

Figure 4. Typical trunk-based versioning adapted from [94,95].

Test

In the test phase of DevOps, automated testing is performed continuously to ensure the quality of the software artifact [84,102]. There are different ways to include test cases such as units and integration while the software is being written. One such method is to use Test-Driven Development (TDD) [103]. In this case, the developer writes the test cases first, and then the actual functionality. There is another approach called Behavior-Driven Development (BDD), which is an extension of TDD [104]. Some good practices, such as code coverage [105], are part of the DevOps pipeline, and some cloud DevOps services, such as Azure DevOps [106], provide this within the service.

Release

Once all test cases are passed, the software is ready for deployment. By this phase, a code change has already passed a series of manual and automated tests [107]. Regular feature releases can be carried out according to a regular schedule or once milestones are met. A manual approval process could be added at the release stage, allowing only a few people within an organization to authorize a release for production [74,78].

Deploy

In the deployment stage, the focus is on continuously deploying or redeploying the software. The deployment varies depending on the type and nature of the application. The deployment process can be automated easily for the production environment, using virtualization or containerization as the orchestration technology. Jenkins is one of the most widely used deployment tools in the industry [53,87].

Operate

The operate phase involves maintaining and troubleshooting applications in a production environment. Teams ensure system reliability and high availability. They aim for zero downtime while reinforcing security [108]. Selecting the proper hardware size for scaling or implementing the scaling methods on the cloud is crucial for the application's uptime and ability to handle high loads [109,110]. This configuration is performed in the operate phase.

Monitor

After the application is deployed and configured to handle the ever-changing load, it is essential to monitor the application to ensure that it is stable enough to handle the promised uptime [111]. The monitor phase helps perform operations such as application health tracking and incident management. "New Relic" is one such tool to monitor the application.

3.1.2. DevOps Pipeline

In practice, some of the DevOps components mentioned above are combined to form a sequence of operations called a pipeline. There is no single generalized pipeline structure [80,112]. Every pipeline is unique, and the pipeline structure depends on the nature of the application and the implementation technology. In DevOps, the following common pipeline components can be found [53,54,58,77,84]:

- Continuous Integration (CI);
- Continuous Delivery (CD);
- Continuous Deployment;
- Continuous Monitoring.

These pipelines include DevOps components mentioned in the previous section. Most of the DevOps implementations consider CI and CD as the core components.

Continuous Integration

Continuous integration is the practice of integrating code changes from multiple developers into a single source via automation [77]. The important practice of CI is that all developers commit the code frequently to the main or trunk branch [113] mentioned in Section 3.1, subsection 'Plan' After the commitment, the code building is performed as explained in Section 3.1, subsection 'Build'. As soon as the build succeeds, the unit test cases are run as explained in Section 3.1, subsection 'Test'. Primarily, CI uses SCM (Software Configuration Management) tools such as Git, as explained in Section 3.1, subsection 'Code', to merge the code changes into SCM for code versioning. CI also performs automated code quality tests, syntax style reviews, and code validation [114]. Since many developers integrate the code, the following issues may occur:

- **Code or merge conflicts:** This is the state when SCM is unable to resolve the difference in code between two commits automatically. Until the merge conflict is resolved, the SCM will usually not allow the code to be merged. Usually, merge conflicts happen if the changes are inconsistent when merging changes from two branches [115].
- **Dead code:** Dead code is a block of code never reached by the execution flow [116]. Dead code can be introduced at any step in the programming as part of a code merge. For instance, consider two feature branches: Feature A and Feature B. Feature A's developers might have considered some exceptions from Feature B and created code blocks to handle that. However, if this exception never happens or some logic in the Feature B branch changes, this exception block will never be executed [117,118].
- **Code overwrites:** As the software evolves, there is a high possibility that the old code needs to be updated to meet the ever-changing requirements. Nevertheless, this might also affect the old features, resulting in code breaks after merging.

One of the primary benefits of CI is that it saves time during the development cycle by identifying and addressing conflicts early [119]. The first step in avoiding the abovementioned issues is to set up an automated testing pipeline [80,120].

In a CI testing pipeline, the code should be built successfully with no code conflicts before the tests are run. The CI pipeline tests vary depending on the nature of the application. To get an early warning of the possible issues, it is suggested to run this CI pipeline for every branch of the SCM and not only on the main or trunk branches. The basic CI pipeline is ready once the SCM and an automated unit testing framework are ready. The following are some of the critical points to keep in mind while implementing CI:

- Unit tests are fast and cost less in terms of code execution time as they check a smaller block of code [121].
- UI tests are slower and more complex to run, more complex to set up, and might also have high costs in terms of execution time and resource usage. Furthermore, a mobile development environment with multiple emulators and environments might add more complexity. Hence, UI tests should be selected more carefully and run less frequently than the unit tests [103,122].
- Running these tests automatically for every code commit is preferred, but doing this on a development or feature branch might be costlier than manual testing [123].
- To meet the code coverage criteria [105], both white-box and black-box testing could be part of CI. However, white-box testing can be time-consuming, depending on the codebase [124].
- Combining code coverage metrics with a test pipeline is the most effective way to know how much code the test suite covers [105]. This coverage report can help in eliminating dead code [125].

The following two practices help to avoid or detect early merge conflicts [115]:

- Pushing of local changes to the SCM should be performed early and often.
- Changes should be pulled from the SCM before pushing. This frequent code pulling will reveal any merge conflicts early.

Various software builds such as Integration builds should be rebuilt and retested after every change [119]. In a CI tool, if every test case runs smoothly without issues, and if there are no merge conflicts, the CI tool shows the current build status as "Pass". If anything goes wrong, this will be changed to "Fail". The priority of the whole development team is to keep the builds "Passing" [126].

Continuous Delivery and Continuous Deployment

Continuous Delivery (CD) and Continuous Deployment are implemented in the DevOps pipeline after CI. In Continuous Delivery, the aim is to keep the application ready for production deployment. At the least, unit test cases, a few optional quality checks, and other tests should have been completed before continuous delivery [54,127]. Continuous Delivery/Deployment is a process of deploying the application to various deployment

instances, such as testing or production. The key differences between Continuous Delivery and Continuous Deployment are as follows: The Continuous Delivery process is the frequent code shipping to production or test instances manually, whereas Continuous Deployment is the automated deployment of code to the instances. In both cases, the code is kept ready for deployment at any point [53,58,120].

Continuous Delivery (CD) is an extension of the previously discussed CI. CD automatically deploys all code changes to the testing and/or production environment after the code-building stage. It is possible to deploy the application manually from the CD pipeline utilizing a manual trigger. CD is mainly used for automated deployments as soon as the build artifacts are ready. In this paper, a build artifact is defined as a binary such as a container or a web portal build after this binary has passed all of the tests and the build stage [128,129]. CD is beneficial when the build artifacts are deployed to production as soon as possible. This frequent deployment ensures that the software release contains small batches, which are easy to debug in case of any issues in the post-production deployment. Figure 5 shows the sample CI/CD pipeline.

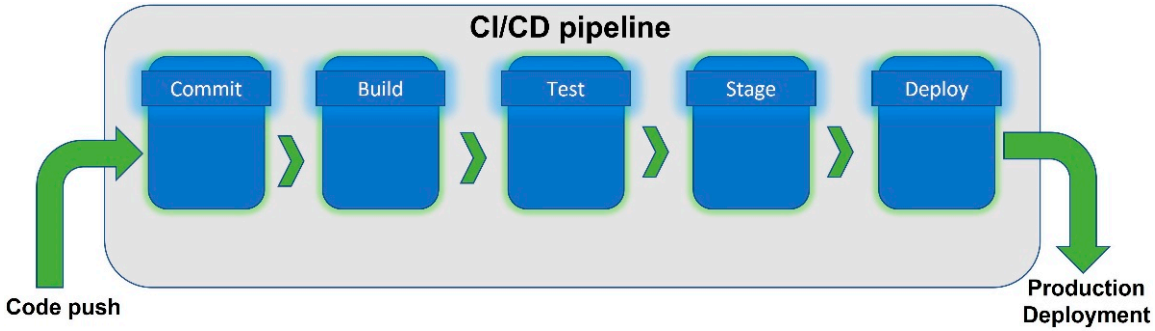

Figure 5. A typical CI/CD pipeline adapted from [54,77,80,127].

Continuous Monitoring

Continuous Monitoring (CM) is an automated DevOps pipeline to monitor the vitals of the deployed application on the production instance. CM comes at the end of the DevOps pipeline and provides real-time data from the monitoring instance. CM helps to avoid and track system downtimes and evaluate application performance, security threats, and compliance concerns [130].

3.2. MLOps

Machine Learning Operations (MLOps) is a set of practices that aims to maintain and deploy Machine Learning code and models with high reliability and efficiency. MLOps is primarily based on DevOps practices such as CI and CD to manage the ML life cycle [131,132]. The main target of MLOps is to achieve faster development and deployment of the ML models with high quality, reproducibility, and end-to-end tracking. Like DevOps, MLOps also enables a shorter code–build–deploy loop and aims to automate and monitor all steps of ML [9,68,69].

3.2.1. MLOps Workflow and Components

Like DevOps, the focus of MLOps is to automate the software delivery process throughout, ensuring continuous delivery and a feedback loop of the software. However, in most cases, the ML application must work with the other software assets in a DevOps-based CI/CD environment. In such cases, additional steps are introduced to the existing DevOps process because existing DevOps tools and pipelines cannot be applied to ML applications [9]. The adaptation of MLOps practices is still in its initial stages, as there is little research on MLOps compared with DevOps [62]. For the ML systems, data scientists and operations teams are trying to automate the end-to-end life cycle of ML by utilizing DevOps

concepts [66]. Due to the variations in ML methodologies, it is challenging to generalize MLOps components.

To date, numerous designs with different components have been proposed, such as Iterative/Incremental processes [132] and Continuous Delivery for Machine Learning (CD4ML) [133]. In 2015, Sculley et al. [4] highlighted that in most real-world ML applications, the quantity of actual ML code is significantly smaller than the surrounding infrastructure, and a vast infrastructure supports this small ML code. The authors also discussed technical issues and challenges in ML systems, such as model complexity, reproducibility of the results, testing, and monitoring. Most of these are also relevant for DevOps components. Hence, it is vital to include the data, infrastructure, and core ML code in the MLOps life cycle. Numerous ML life-cycle processes such as CRISP-ML(Q) [134] have been proposed to establish a standard process model for ML development. John et al. [62] presented a maturity model outlining various stages by which companies evolve their MLOps approaches.

A generic MLOps workflow is shown in Figure 6. As mentioned, the concerns cited by Sculley et al. [4]—such as code maintenance problems and system-level issues—are also present in traditional software development, and DevOps solves most of them via CI/CD. This CI/CD process creates reliable pipelines with assured quality to release the software into production. A cross-functional team is a way of involving expertise from different functional areas [78,135], such as data scientists and ML engineers [131]. In MLOps, a cross-functional team produces ML applications in small increments based on three parameters: code, data, and models. These can be released and reproduced at any time, using a constant seed value for random sample initialization to set the weights of the trainable layers, if applicable [136]. The following subsections define the generic MLOps process model illustrated in Figure 6.

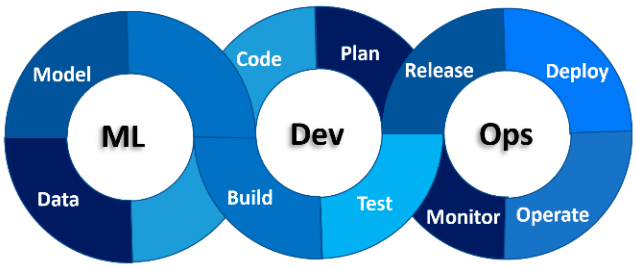

Figure 6. Typical MLOps workflow adapted from [68,131].

The MLOPs workflow in Figure 6 is similar to the DevOps workflow in Figure 2 but introduces two new components: data and model. Furthermore, the MLOps components testing, deployment, and monitoring are slightly different from their DevOps counterparts. Along with data and model, these differences are explained in the following sections.

Data

ML is driven by data; hence, data analysis and operations are vital to MLOps. [137,138]. Unlike DevOps, ML operations are experimental, which is true in almost all of the steps of MLOps. For instance, hyperparameter optimization is varied during implementation. The same is true for data, and the following operations are involved in data analysis [139,140]:

- Data extraction;
- Data validation;
- Data analysis;
- Data preparation.

The sequence and usage of these components depend on the type of ML and the nature of the application.

Data extraction is mainly concerned with the gathering of data from different sources. The data sources—such as online APIs, cloud-based data lakes, CSV files, or a combination

of these—can be diverse. Extra precautions should be taken while extracting the data for some ML tasks. For instance, for a classification issue, the data must be balanced after they are extracted. Failing to do so might degrade the classifier's performance [141]. The data extraction component in the MLOps pipeline is usually the first step, in which data from one or more sources are integrated for further processing [139].

To detect inaccurate data early and avoid training ML models with flawed data, the suggested technique is to incorporate a **data validation** process [140]. Common data quality issues include the following [140,142]:

- Incomplete data; for instance, the presence of null values.
- Inconsistent data, such as issues with the data type.
- Inaccurate data; for example, data collection with the wrong measurements.

Data analysis is a crucial step in the creation of a model. In this step, Exploratory Data Analysis (EDA) is performed to understand the characteristics of the data [138]. Depending on this knowledge, feature engineering is performed in the following steps, and a suitable model is designed.

In **data preparation**, the validated data are split into three standard datasets: training, test, and validation [143]. Features are selected from the data, data-cleaning operations are performed, and some extra features are added after EDA. EDA shows the trends and patterns in the data, and we can add more features from new data sources to support the existing data [144]. In data preparation, common data quality issues can be fixed. If required, data transformations such as date–time format matches from different data sources [145] are performed on the data and, finally, the three sets of data are sent to the model.

Model

The ML model is the heart of any ML application. Neural networks are the most common type of ML model, and the rest of this paper assumes that the ML model is a neural network. Once the model structure is defined, model training, model evaluation, and model validation operations are performed.

Model training trains one or several models with the prepared data. Hyperparameter optimization is performed, where model variables such as the number of layers and nodes in each layer are optimized in different iterations. After this optimization, the model is trained or well-fitted [146,147].

Once the model is trained, it is evaluated in **model evaluation** of the validation data. The trained model is evaluated using the held-out validation datasets to measure the model's quality. Model validation gives the measure of performance of the model [148]. Metrics such as absolute error and mean absolute error are used to define the model quality. These metrics are helpful in testing and comparing different models [149,150].

Testing

As in DevOps, unit and integration tests should be performed. Testing of ML models is mostly limited to checks related to the convergence of models, shapes of passed tensors/vectors, and other model-related variables. However, there is a lot of code surrounding an ML model, which should also be tested [151]. However, white-box testing for ML-based systems could entail high test efforts due to the large test input space [152]. In MLOps, test cases should check the proper input and output data format.

Deployment

Unlike DevOps, deploying ML applications is not straightforward, especially if the ML application is a part of a DevOps application such as a web API. In ML applications, the arrival of new data triggers the retraining and redeployment of models. An automated pipeline must be created to perform these actions [153].

Monitoring

It is crucial to monitor the performance of the deployed ML model. Continuous monitoring helps to understand the model performance and trigger retraining if required [154]. In DevOps, the main concern is ensuring that the application is healthy and able to handle the load. If the ML is part of an application such as a web API, the monitoring component should check for ML parameters such as data drift and model performance [131,155].

As in DevOps, some of these steps are combined to form a pipeline. Usually, these are combined with DevOps pipelines. As in DevOps, there is no single generalized pipeline structure. Every pipeline is unique, and the pipeline structure depends on the nature of the application, the type of ML, and the implementation technology.

3.2.2. MLOps Pipeline

In MLOps, the following standard pipeline components are implemented [62,69,131]:

- Continuous Integration (CI);
- Continuous Deployment (CD);
- Continuous Training (CT).

Continuous Integration

As in DevOps, the CI pipeline is about the testing and validation of code components. For ML applications, data and model validations are added along with classical unit and integration tests [65]. Unit tests are written to cover the changes in feature engineering, and different methods used to implement the models. Moreover, tests should be written to check the convergence of the model training. During training, a machine learning model reaches a convergence state when the model loss value settles within an error range, after which any additional training might not improve the model's accuracy [156].

Continuous Deployment

There are considerable changes in the Continuous Deployment pipeline compared to DevOps. As the ML models evolve continuously, verifying the models' compatibility with the target deployment environments with respect to computing power and any changes in the deployment environments is essential. The process changes depending on the use case and whether the ML prediction is online or batch processing [157].

Continuous Training

This new pipeline component is unique to MLOps., The Continuous Training (CT) pipeline automatically retrains the model [158]. Different ML components explained in Section 3.2.1 are automated to work in a sequence to achieve this. Retraining the model is essential, as the data keep changing or updating in any ML application. To cope with the new incoming data, the model needs retraining. This retraining includes automating several model retraining, data validation, and model validation processes. To initiate such processes, triggers are included in the pipeline.

Some additional components, such as feature storage and metadata management, are used along with these pipelines in MLOps. They help in managing data and reproducibility aspects. These are discussed below.

3.2.3. Feature Store

Due to many variations, such as the components involved in the ML applications, feature stores are used. This feature store acts as a central repository for standardizing the definition, access, and storage of a feature set. A feature store helps to achieve the following [159]:

- Store commonly used features;
- Build feature sets from the available raw data;
- Reuse custom feature sets;

- Model monitoring and data drift detection;
- Transform and store the data for training or inference purposes.

3.2.4. Metadata Management

The fundamental nature of ML is experimentation. It would not be easy to track the steps that lead to an ideal model or the best-performing dataset with many experiments in all of the different components. With the help of metadata management, almost all of the metadata for the whole ML process can be tracked and used to repeat the desired result. The most significant metadata include [160]:

- Experiments and training metadata—Metadata such as environment configuration, hardware metrics, code versions, and hyperparameters.
- Artifact metadata—Metadata such as dataset paths, model hashes, dataset previews, and artifact descriptions.
- Model metadata—Model-related metadata such as model versions, data and performance drifts, and hardware usage.
- Pipeline metadata—Pipeline metadata such as node information and completion reports of each pipeline.

MLOps pipelines might look highly automated. Nevertheless, not all of the pipelines or pipeline components are necessary to implement. The pipeline and its components can be selected depending on the implementation and use case. Sculley et al. [4] mentioned that only a minority of the application is ML code, since most of the system is composed of data collection and verification components, model analysis and building, resource and metadata management, automation, and configuration. Google Cloud [161] proposes three MLOps process levels for implementing the CI/CD pipeline for different needs based on the work of Sculley et al. [4]; these define the different levels of maturity of the MLOps processes:

- MLOps level 0: Manual process;
- MLOps level 1: ML pipeline automation;
- MLOps level 2: CI/CD pipeline automation.

3.3. Research Gap

In the next section, MLOps is applied for a time-series forecasting application, which predicts the price of a day-ahead hourly electricity market. The MLOps level 2 automated CI/CD pipeline is implemented. CI and CD services are utilized for reliable delivery of the ML results.

This use case fills the following research gap observed in Sections 2 and 3:

- Even though the core ML is a tiny part of the whole software ecosystem, the ML application needs various new tools for the MLOps implementation.
- MLOps tools might not be compatible with the DevOps tools, burdening the complete system.
- Creating an MLOps pipeline with traditional software—such as a web application where DevOps is already implemented—is an issue.
- Additionally, to explain the generalization of the created MLOps pipeline and the tools, Case Study 1 is extended to two more case studies. In the last two case studies, the price forecast solution (Case Study 1) is adapted with minimal work.

4. Case Study 1: Forecasting an Hourly Day-Ahead Electricity Market Using an MLOps Pipeline

The reliable operation of the electric power grid relies on a complex system of markets, which ensures that electric power consumption and generation are matched at every point in time. This matching is crucial for the stability of the grid, since the grid cannot store electric energy. From an MLOps perspective, forecasting the price of any electricity market depends on the market schedule, and the ML experts do not need to understand how the

market contributes to stabilizing the grid. The timing characteristics of the market represent important knowledge for the developers of time-series forecasting solutions. In particular, two characteristics are crucial:

- Does the price change weekly, daily, hourly, or at some other interval?
- When does the market participation occur and, thus, how far into the future should forecasts be available?

There are numerous markets in a single country, and significant differences can exist between different countries. Thus, an ML team working on electricity price forecasting would do well to avoid hardcoding assumptions related to the questions above, and should instead parameterize them. A common market structure in Europe and elsewhere is the hourly day-ahead market [162,163], which used as a concrete example in this paper. Such markets have the following characteristics:

- The market interval is one hour; in other words, there is a separate price for each hour.
- The market participants need to place their bids on the previous day; for example, before the market deadline today, separate bids should be placed for each hour of the next day.

In our case study, we look at a specific hourly day-ahead market—the Finnish Frequency Containment Reserves for Normal Operations (FCR-N) market. The FCR markets compensate participants for maintaining a reserve that can be activated to generate or consume energy in case such an activation is required due to a momentary imbalance in the power grid. An offline neural-network-based forecasting solution for this market is presented in [164].

4.1. The Frequency Containment Reserves Market

With the advent of smart grids and Virtual Power Plants (VPPs), various Distributed Energy Resources (DERs)—such as smart loads, batteries, photovoltaics (PVs), and wind power—are being exploited on various electricity markets, including frequency reserves [165–168]. Frequency reserves with a fast response time for frequency deviations are generally called Primary Frequency Reserves (PFRs), and traditionally they consist of fossil-fuel-burning spinning reserves. These reserves are now being replaced with DERs in the push towards reducing carbon emissions [169]; due to this, under the current allowed delays for PFRs, the reduced grid inertia is becoming a threat to the stability of power systems [170]. The FCR-N market was selected among other ancillary services for this case study, as the FCR-N does not have hard real-time constraints and a minimum power bid for market participation. The FCR-N is a day-ahead market, and bidders must submit all of the bids for the hours of the next day before 6 p.m. of the current day [171,172].

If the day-ahead reserve market prices can be predicted, then the DER owners can anticipate variations in price peaks and low or zero prices. Thus, this case study presents a solution based on artificial neural networks, which are deployed online, so that the predictions are automatically updated and available before the bidding deadline of the day-ahead market. A transformer-based ANN model is exploited to predict the day-ahead ancillary energy market prices. The MLOps pipelines are configured to ingest the data at 1 p.m. on the current day, so the forecasts are available, e.g., at 2 p.m., so that the person or system doing the bidding has the forecast a few hours before submitting the bid.

4.2. Prediction Model

For the energy price predictions in ML, ANNs are widely used. For instance, based on the architecture defined in [173], Recurrent Neural Networks (RNNs) and feed-forward neural networks are the two major ANN categories. RNNs can predict the high energy spikes better, whereas feed-forward networks can predict the spot market prices for day-ahead prediction [174]. An ANN was employed to predict ancillary market prices by considering different data sources where the results outperformed Support-Vector Regression (SVR) and Autoregressive Integrated Moving Average (ARIMA) [164]. For implementing

the FCR-N market price forecasting using the MLOps pipeline, this case study uses the Temporal Fusion Transformer (TFT) model structure defined in [175]. Energy price prediction datasets have a time component, and forecasting the future price values can provide significant value for multi-horizon forecasting, i.e., predicting variables of interest at multiple future time steps. Deep Neural Networks (DNNs) have been used in multi-horizon forecasting, showing substantial performance improvements over traditional time-series models. However, most existing RNN models often do not consider the different inputs commonly present in multi-horizon forecasting, and either assume that all exogenous inputs are known in the future or do not consider static covariates. Conventional time-series models are influenced by complex nonlinear interactions between many parameters, making it difficult to explain how such models arrive at their predictions. Attention-based models are proposed for sequential data such as energy price prediction datasets. However, multi-horizon forecasting has many different types of inputs, and attention-based models can provide an understanding of appropriate time steps, but they cannot contrast the importance of different features at a given time step, and TFT solves these issues in terms of accuracy and interpretability.

TFT is an attention-based architecture that combines multi-horizon forecasting with interpretable insights into temporal dynamics. TFT utilizes recurrent layers for local processing and interpretable self-attention layers for learning long-term dependencies with the knowledge of temporal relationships at diverse scales [175]. The major components of TFT are summarized below.

- *Gating mechanisms* skip any unused components of the model.
- *Variable selection networks* choose relevant input variables at each time step.
- Static features can have a meaningful impact on forecasts, and *static covariate encoders* integrate such features based on which temporal dynamics are modeled.
- A sequence-to-sequence *temporal processing* layer to learn long-term and short-term temporal relationships.

Table 2 provides additional information on TFT hyperparameters for practitioners who may wish to reproduce the results.

Table 2. TFT hyperparameters.

Temporal Fusion Transformer	
hidden_size	32
attention_head_size	1
dropout	0.1
hidden_continuous_size	16
learning_rate	0.03
loss	QuantileLoss()
log_interval	2
reduce_on_plateau_patience	4
Trainer	
gpus	1
max_epochs	10
gradient_clip_val	0.1

4.3. MLOps Pipeline for FCR-N Market Price Forecasting

The main scope of this example is to define an MLOps pipeline for such applications and determine how the whole process could be automated. First, the data are ingested via Fingrid and the Finnish Meteorological Institute's (FMI) REST API into raw data storage. The data from Fingrid and FMI are available online via the REST API, but not

as a manual download via files; hence, it is possible to automate this pipeline. Next, the data are prepared for further processing by selecting the essential features from the raw data storage, and this feature selection is based on past experimentation. Data are then validated for missing values or the presence of *NaN* values and stored in a feature store for data reusability. Additionally, EDA is performed on the data to understand any data drifts. The next step is to build the model through model training and evaluation. Once the model is evaluated, the prediction is performed on the new dataset. The performance of the model is evaluated for monitoring purposes. The resulting pipeline architecture is shown in Figure 7.

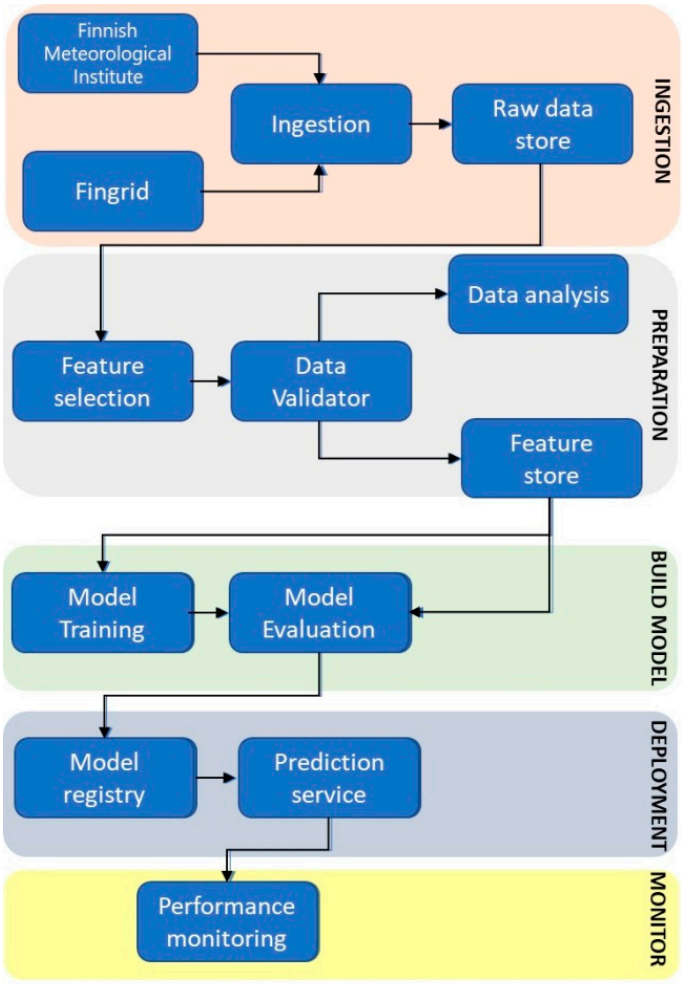

Figure 7. Proposed MLOps pipeline.

4.4. MLOps Runtime Environment

As mentioned in Section 3.1, subsection 'Code', the first step is to ensure that the code is appropriately versioned. GIT was selected as the version control for this implementation, and GIT feature branching was followed. Figure 8 shows the commits and GIT branches.

Two feature branches were created for the development of the solution, namely, "feature/build-model" and "feature/ingest-data". Additionally, one "develop" branch and one "release" branch were created. The develop branch was branched out from the main branch, and the feature branches were branched out of the develop branch. The feature branches were frequently merged back to the development branch, and all the feature branches were updated with the recent changes. Once the final code was merged back to the develop branch, the code was planned for release from the release branch. The code was versioned correctly, and a stable version of the code was always kept in the main branch after tagging correctly.

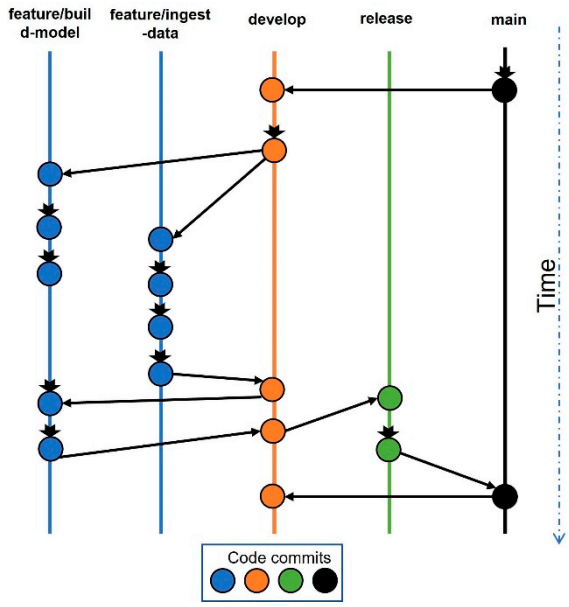

Figure 8. Commits in GIT showing branches, merging, and tags.

In this use case—MLOps level 2—CI/CD implementation was followed as explained at the end of Section 3. Hence the CI/CD should include source control, ML pipelines, test services, model services, feature storage, and deployment services. The first step in the CI/CD process is to create pipelines. These pipelines run in a sequence to implement the price forecast task. Moreover, the pipelines should provide a supportive environment, including runtimes and libraries. We selected PyTorch as the ML library and Python as the scripting language. Python libraries such as NumPy and Pandas were installed in a virtual environment and loaded by the pipeline at the beginning to prepare the environment for the forecasting job. For this implementation, 10 pipelines were selected, and the modular code was created to run in each component. The required unit tests were also implemented and included along with data validation. The trigger was set to any commit changes in the SCM main branch; the pipelines would be triggered upon such changes, and the code would be deployed to generate the forecasting results. There is a variety of software available for creating CI/CD pipelines. Jenkins was selected as the CI/CD server because it is the most commonly used CI/CD server for DevOps. MLOps pipelines were incorporated using Jenkinsfile and Python. In this use case, we also used the DevOps process for implementing a web UI using Django, where the Django web framework was used for viewing the predicted result. All of the required software—including GIT, Jenkins, Python, Anaconda, and Docker—was installed on the CSC cloud.

Figure 9 shows the pipelines created for the FCR-N market price forecaster. A new SCM commit change at the main branch would trigger the pipeline. The following pipelines were created in this example implementation:

- Data ingestion;
- Data preparation;
- EDA;
- Model building;
- Deployment and monitoring pipelines with a few sub-pipelines.

"Data Ingestion—Data Fetch" fetches the data from the Fingrid and FMI API. "Data Ingestion—Store Raw Data" pipeline stores these data in an SQL database, and PostgreSQL is used as the database, which stores the raw data. "Data Preparation—Feature Selection" performs the feature selection from the raw data. Different experiments have been performed in the past, based which this pipeline selects the features. "Data Preparation—Data Validator" validates the data and checks against different validation rules and data consistency, including null values, the presence of NaN, and data types. Along with this

code level, unit test cases are also executed to ensure code integrity. These data are then stored in a feature store for future use and analysis using the "Data Preparation—Feature Store" pipeline.

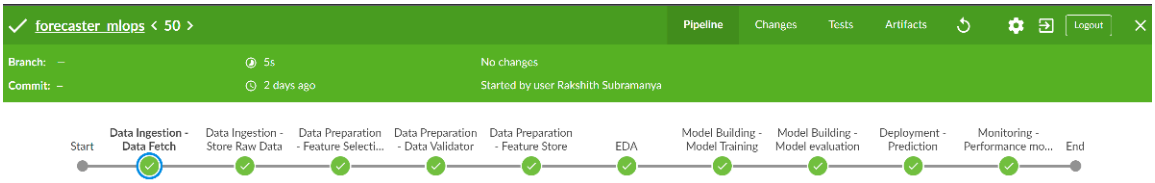

Figure 9. MLOps pipeline for FCR-N market price forecasting.

Data analysis is performed automatically with the "EDA" pipeline to avoid data drift and other issues with data quality. The next step is to normalize the data and build the model, and this is done via the "Model Building—Model Training" and "Model Building—Model evaluation" pipelines. It is then necessary to create a docker container of the prediction model. "Deployment—Prediction" pipeline does that, and then forecasts the FCR-N market price for the next 24 h. The forecasted result is stored in the PostgreSQL database. The model's accuracy and performance are monitored via the "Monitoring—Performance monitor" pipeline. The EDA and performance monitoring results are monitored regularly, and the pipelines are updated if the performance has gone down.

Due to the modular nature of the CI/CD pipeline design, it is easy to plug in/out a module. If a build fails—for instance, at the "Data Ingestion—Data Fetch"—the CI/CD system shows a detailed log on the CI/CD UI, as shown in Figure 10.

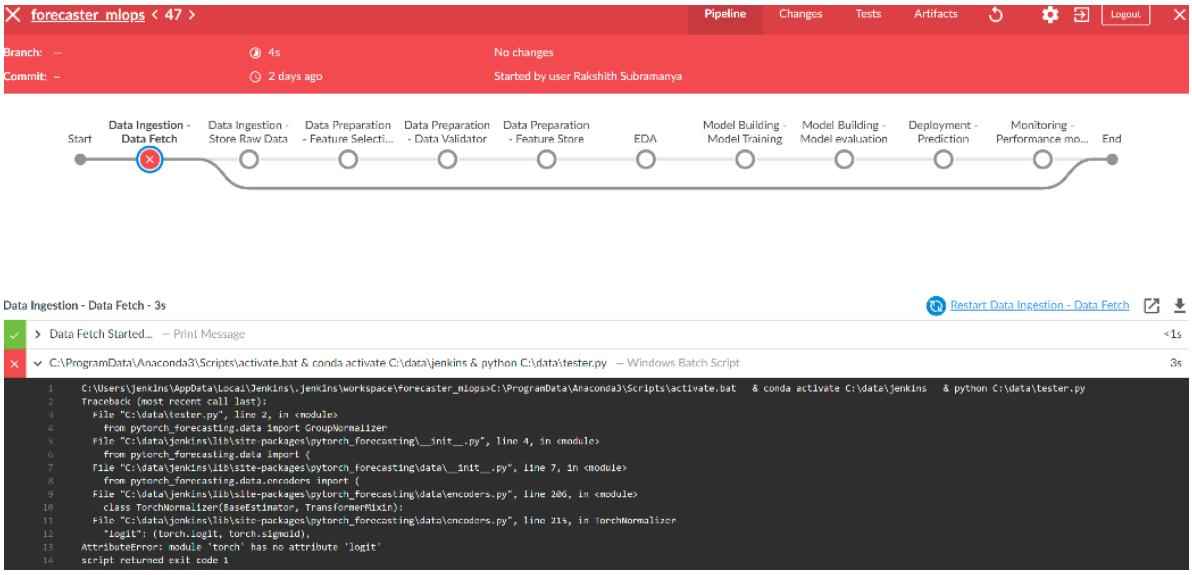

Figure 10. Build fail notification.

These logs are maintained in the CI/CD system so that it is easy to debug and replicate the errors in the developer's machine using the local build discussed in Section 3.1, subsection 'Build'. Figure 10 shows the failure pipeline. Once the pipeline finishes the job execution, depending on the build's status, an email is triggered from the CI/CD system to the configured emails notifying about the new build job. This notification system is created while building the pipelines.

MLflow was selected as the ML application life-cycle management tool. MLflow was integrated with the ML model operations to track the model parameters and performance metrics. Figure 11 shows the MLflow UI hosted on an SaaS service.

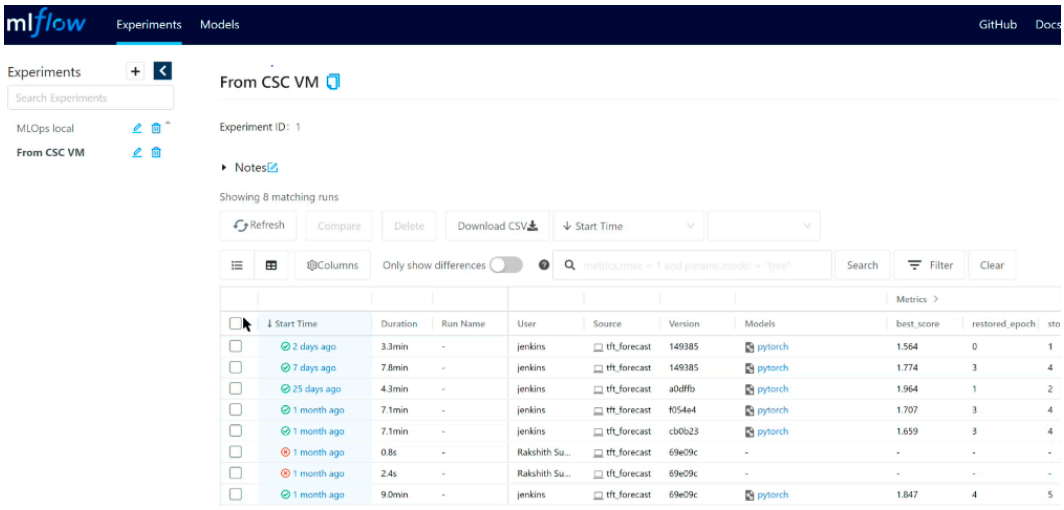

Figure 11. ML life-cycle management with MLflow.

MLflow includes experimentation tracking and reproducibility. This is achieved by logging the metrics and the parameters of each experiment. As shown in Figure 11, several experiments' results can be logged and compared. MLflow stores the artifacts such as configuration files that save information about the model input and the model under training. These artifacts can be stored in a cloud object store such as MinIO for future reproducibility, as shown in Figure 12.

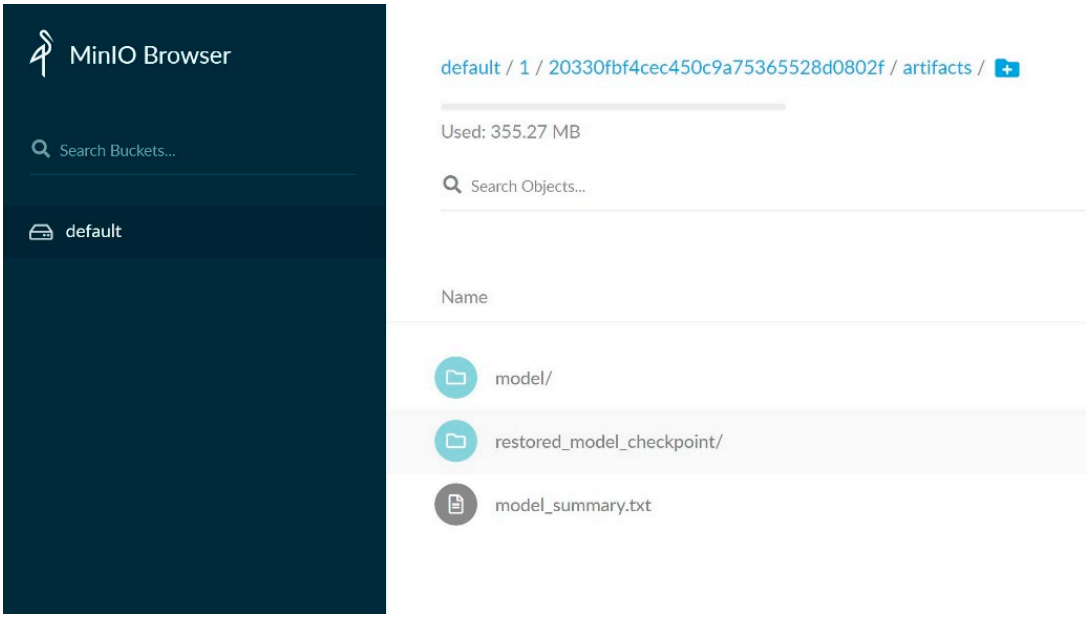

Figure 12. ML artifact storage using the MinIO object store.

4.5. FCR-N Forecasting Results

Figure 13 shows the one-day predicted vs. actual price prediction for the FCR-N market price. The ML model runs daily, and the predicted values are stored in the database. A Django-based web UI was designed for viewing these values and hosted in the CSC cloud as an SaaS, as shown in Figure 14. This SaaS service can also act as a REST API to expose the predicted data to a third-party system.

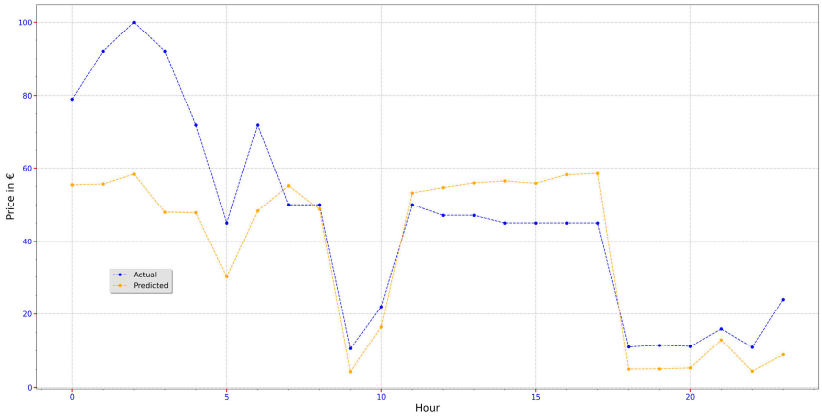

Figure 13. Predicted vs. actual price for 16 April 2020.

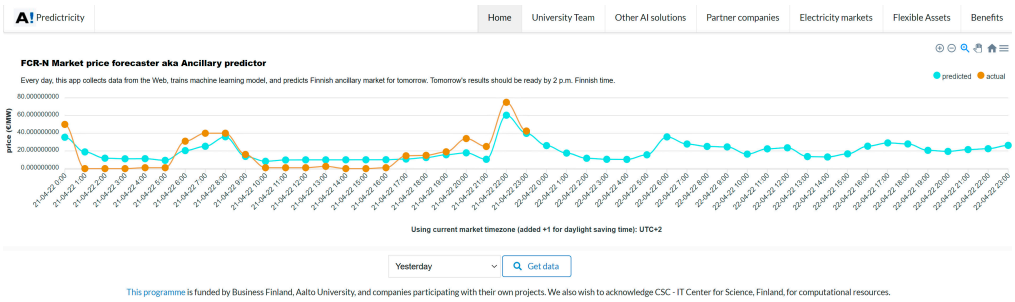

Figure 14. Web UI for exploring the model prediction values.

5. Case Study 2: National Electricity Consumption Forecast Adapted from the Price Forecast MLOps Pipeline

In the above use case, the MLOps pipeline was defined and implemented as shown in Figure 7, while in this use case, the same MLOps pipeline was adapted with minimal changes to forecast the Finnish national electricity consumption. The data were collected from the Finnish Transmission System Operator (TSO) via an online API. The TSO defines the electricity consumption as follows:

$$\text{Consumption} = \text{Production} + \text{Import} - \text{Export} \quad (1)$$

This use case was implemented with all of the pipeline components defined in Figure 7. However, the following changes were made to the previous MLOps source code:

- In the TFT, *the FCR_N price* variable was replaced with *electricity_consumption* as the prediction variable.
- In the GIT, a new feature branch was created.
- A new experiment name was created for tracking the ML model parameters using MLflow.
- A new Jenkins project as created, and MLOps pipelines were incorporated using the same Jenkinsfile.
- Unit test cases are updated.
- To explore the predicted values via web UI, as shown in Figure 14, the corresponding legend and variable names were changed.

Except for the test cases, most of the abovementioned changes were in the configuration files and, most importantly, the same MLOps pipeline and tools were reused on the same cloud platform. Figure 15 shows the one-day predicted vs. actual electricity consumption forecasts for Finland.

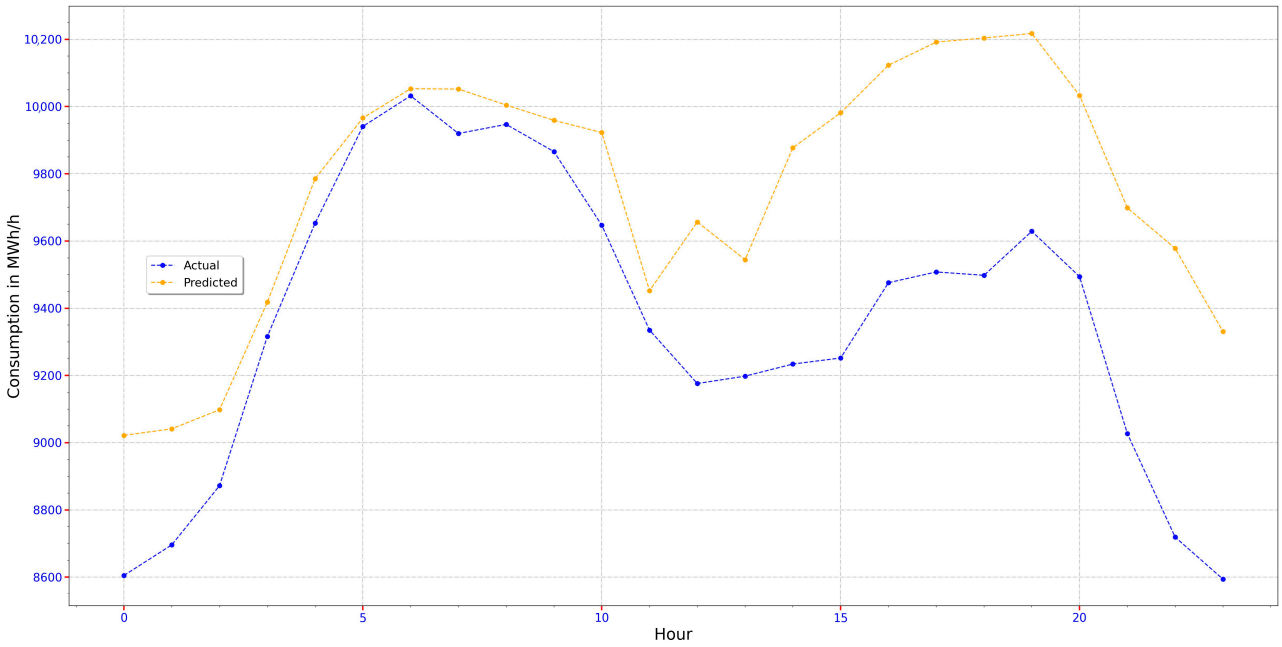

Figure 15. Predicted vs. actual electricity consumption for 16 April 2020.

6. Case Study 3: National Electricity Generation Forecast Adapted from the Price Forecast MLOps Pipeline

In the previous use case, electricity consumption was forecasted by adapting to the MLOps pipelines created for FCR-N market price forecasting. In this example, electricity production was forecasted, and the MLOps pipeline changes made for the previous use case were used here. As in the electricity consumption forecasting, the data were collected from the Finnish TSO via an online API.

However, the following changes were made to the previous MLOps source code:

- In the TFT, the *electricity_consumption* variable was replaced with *electricity_production* as the prediction variable.
- In the GIT, a new feature branch was created.
- A new experiment name was created for tracking the ML model parameters using MLflow. Creating a new experiment name is helpful in grouping the new experiments within a project or use case.
- A new Jenkins project was created, and MLOps pipelines were incorporated using the Jenkinsfile used for the above use case.
- Unit test cases were updated.
- To explore the predicted values via web UI, as shown in Figure 14, corresponding legend and variable names were changed.

Figure 16 shows the one-day predicted vs. actual electricity production forecasts for Finland.

Case Studies 2 and 3 were implemented with minimal work by adapting to the implementation of Case Study 1. Case Studies 1 and 2 both used the same MLOps pipeline structure and the same set of tools. This generalization is applicable while forecasting similar variables—for instance, wind power forecasting or photovoltaic power generation forecasting—with 1 h as the sampling interval. However, if the sampling interval changes—for example, to 15 min—considerable work must be performed in the MLOps pipeline, such as “Data ingestion” and “Prediction service”.

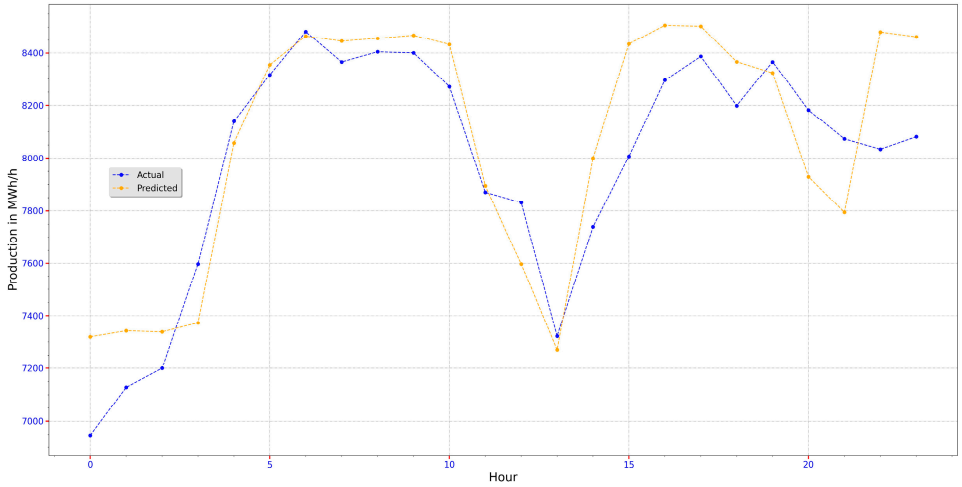

Figure 16. Predicted vs. actual electricity production for 16 April 2020.

7. Conclusions

The main aim of MLOps is to introduce ML products to production by avoiding Development and Operations bottlenecks and automating the workflows. The MLOps system and workflow design need to be modular to accommodate such a system. Such a modular design cannot be generalized and must be specific to the application. This would ensure a system with reduced development, deployment, and monitoring issues. The end-to-end life-cycle management of MLOps is easy. This paper presents a use case where we built modular pipelines for an ML time-series forecasting system. There are several ways to implement MLOps, but the principles of DevOps and modularity should be considered as the primary key factors. Even though it is hard to generalize the pipeline structure, in this paper we explain the things to consider when creating the MLOps design architecture.

The case study is generalizable with minor changes to other hourly day-ahead electricity markets. The main changes are related to identifying the relevant features. The case study is further generalizable to other electricity markets operating on a similar timescale. For example, the market interval in some countries is 30 min or 15 min instead of an hour. For a day-ahead market, the market interval will impact the size of the input layer as well as the output layer of the ANN. For electricity markets that are not day-ahead—for example, intraday markets—the bidding deadlines are different, which needs to be taken into account in scheduling the execution of the deployed containers. All of these generalizations require minor efforts in the form of manual work. A topic for further research would be the development of a generic MLOps solution for day-ahead or intraday electricity markets, which would further reduce this manual work. The current case studies are only integrated with the web application to explore the forecasted values. This MLOps work could be extended for Virtual Power Plants, which use the predicted data for managing the assets using the REST API, where concurrency and response time matter. Currently, the REST API implements a basic authentication for exposing the predicted values, and this basic authentication could be further improved.

Author Contributions: Conceptualization, R.S. and S.S.; methodology, R.S. and S.S.; software, R.S.; validation, R.S. and S.S.; formal analysis, R.S. and S.S.; investigation, R.S.; resources, R.S. and S.S.; data curation, R.S.; writing—original draft preparation, R.S. and S.S.; writing—review and editing, R.S. and S.S.; visualization, R.S.; supervision, S.S. and V.V.; project administration, S.S.; funding acquisition, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by Business Finland under grant 7439/31/2018.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The calculations presented above were performed using computer resources within the Aalto University School of Science's "Science-IT" project and the CSC—IT Center for Science, Finland, for computational resources.

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

References

1. Pang, C.; Hindle, A.; Barbosa, D. Understanding DevOps education with Grounded theory. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, Seul, Korea, 5–11 October 2020; ACM: New York, NY, USA, 2020; pp. 260–261.
2. Macarthy, R.W.; Bass, J.M. An Empirical Taxonomy of DevOps in Practice. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portorož, Slovenia, 26–28 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 221–228.
3. Wahaballa, A.; Wahballa, O.; Abdellatief, M.; Xiong, H.; Qin, Z. Toward unified DevOps model. In Proceedings of the 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 23–25 September 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 211–214.
4. 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 Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
5. Fitzgerald, B.; Stol, K.-J. Continuous software engineering: A roadmap and agenda. *J. Syst. Softw.* **2017**, *123*, 176–189. [CrossRef]
6. Haindl, P.; Plosch, R. Towards Continuous Quality: Measuring and Evaluating Feature-Dependent Non-Functional Requirements in DevOps. In Proceedings of the 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), Hamburg, Germany, 25–29 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 91–94.
7. Steffens, A.; Lichter, H.; Döring, J.S. Designing a next-generation continuous software delivery system. In Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering, Gothenburg, Sweden, 29 May 2018; ACM: New York, NY, USA, 2018; pp. 1–7.
8. Liu, Y.; Ling, Z.; Huo, B.; Wang, B.; Chen, T.; Mouine, E. Building A Platform for Machine Learning Operations from Open Source Frameworks. *IFAC-PapersOnLine* **2020**, *53*, 704–709. [CrossRef]
9. Makinen, S.; Skogstrom, H.; Laaksonen, E.; Mikkonen, T. Who needs MLOps: What data scientists seek to accomplish and how can MLOps help? In Proceedings of the 2021 IEEE/ACM 1st Workshop on AI Engineering—Software Engineering for AI, WAIN, Madrid, Spain, 30–31 May 2021.
10. Khan, M.A.; Parveen, A.; Sadiq, M. A method for the selection of software development life cycle models using analytic hierarchy process. In Proceedings of the 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India, 7–8 February 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 534–540.
11. Ben-Zahia, M.A.; Jaluta, I. Criteria for selecting software development models. In Proceedings of the 2014 Global Summit on Computer & Information Technology (GSCIT), Sousse, Tunisia, 14-16 June 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 1–6.
12. Öztürk, V. Selection of appropriate software development life cycle using fuzzy logic. *J. Intell. Fuzzy Syst.* **2013**, *25*, 797–810. [CrossRef]
13. Lekh, R. Pooja Exhaustive study of SDLC phases and their best praxctices to create CDP model for process improvement. In Proceedings of the 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India, 19–20 March 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 997–1003.
14. Dayal Chauhan, B.; Rana, A.; Sharma, N.K. Impact of development methodology on cost & risk for development projects. In Proceedings of the 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 20–22 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 267–272.
15. Akbar, M.A.; Sang, J.; Khan, A.A.; Shafiq, M.; Hussain, S.; Hu, H.; Elahi, M.; Xiang, H. Improving the Quality of Software Development Process by Introducing a New Methodology–AZ-Model. *IEEE Access* **2018**, *6*, 4811–4823. [CrossRef]
16. Akbar, M.A.; Sang, J.; Khan, A.A.; Amin, F.-E.; Hussain, S.; Sohail, M.K.; Xiang, H.; Cai, B. Statistical Analysis of the Effects of Heavyweight and Lightweight Methodologies on the Six-Pointed Star Model. *IEEE Access* **2018**, *6*, 8066–8079. [CrossRef]
17. Cho, J. A Hybrid Software Development Method For Large-Scale Projects: Rational Unified Process With Scrum. *Issues Inf. Syst.* **2009**, *10*, 340–348. [CrossRef]
18. Velmourougan, S.; Dhavachelvan, P.; Baskaran, R.; Ravikumar, B. Software development Life cycle model to build software applications with usability. In Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 24–27 September 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 271–276.
19. Fisher, K.G.; Bankston, A. From Cradle to Sprint: Creating a Full-Lifecycle Request Pipeline at Nationwide Insurance. In Proceedings of the 2009 Agile Conference, Chicago, IL, USA, 24–28 August 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 223–228.
20. Poort, E.R. Driving Agile Architecting with Cost and Risk. *IEEE Softw.* **2014**, *31*, 20–23. [CrossRef]

21. Owais, M.; Ramakishore, R. Effort, duration and cost estimation in agile software development. In Proceedings of the 2016 Ninth International Conference on Contemporary Computing (IC3), Noida, India, 11–13 August 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–5.
22. Webster, C.; Shi, N.; Smith, I.S. Delivering software into NASA's Mission Control Center using agile development techniques. In Proceedings of the 2012 IEEE Aerospace Conference, Big Sky, MT, USA, 3–10 March 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–7.
23. Cleland-Huang, J. Safety Stories in Agile Development. *IEEE Softw.* **2017**, *34*, 16–19. [CrossRef]
24. Alyahya, S.; Bin-Hezam, R.; Maddeh, M. Supporting Remote Customer Involvement in Distributed Agile Development: A Coordination Approach. *IEEE Trans. Eng. Manag.* **2022**, 1–14. [CrossRef]
25. Kumar, G.; Bhatia, P.K. Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies. In Proceedings of the 2014 Fourth International Conference on Advanced Computing & Communication Technologies, Rohtak, India, 8–9 February 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 189–196.
26. Sinha, A.; Das, P. Agile Methodology Vs. Traditional Waterfall SDLC: A case study on Quality Assurance process in Software Industry. In Proceedings of the 2021 5th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), Kolkata, India, 4–5 May 2018; IEEE: Piscataway, NJ, USA, 2021; pp. 1–4.
27. Ahmed, A.; Ahmad, S.; Ehsan, N.; Mirza, E.; Sarwar, S.Z. Agile software development: Impact on productivity and quality. In Proceedings of the 2010 IEEE International Conference on Management of Innovation & Technology, Singapore, 2–5 June 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 287–291.
28. Tommy, R.; Mhaisekar, M.; Kallepally, S.; Varghese, L.; Ahmed, S.; Somaraju, M.D. Dynamic quality control in agile methodology for improving the quality. In Proceedings of the 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), Bhubaneshwar, India, 2–3 November 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 233–236.
29. Darrin, M.A.G.; Devereux, W.S. The Agile Manifesto, design thinking and systems engineering. In Proceedings of the 2017 Annual IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 24–27 April 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.
30. Boehm, B. Get ready for agile methods, with care. *Computer* **2002**, *35*, 64–69. [CrossRef]
31. Dybå, T.; Dingsøyr, T. Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.* **2008**, *50*, 833–859. [CrossRef]
32. Samarawickrama, S.S.; Perera, I. Continuous scrum: A framework to enhance scrum with DevOps. In Proceedings of the 2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, 6–9 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–7.
33. Miller, A. A hundred days of continuous integration. In Proceedings of the Agile 2008 Conference, Toronto, ON, Canada, 4–8 August 2008; Microsoft Corporation: Albuquerque, NM, USA, 2008; pp. 289–293.
34. Arachchi, S.A.I.B.S.; Perera, I. Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. In Proceedings of the 2018 Moratuwa Engineering Research Conference (MERCon), Moratuwa, Sri Lanka, 28–30 July 2020; IEEE: Piscataway, NJ, USA, 2018; pp. 156–161.
35. Süß, J.G.; Swift, S.; Escott, E. Using DevOps toolchains in Agile model-driven engineering. *Softw. Syst. Model.* **2022**, 1–16. [CrossRef]
36. Nagarajan, A.D.; Overbeek, S.J. A DevOps Implementation Framework for Large Agile-Based Financial Organizations. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*; Springer International Publishing: Cham, Switzerland, 2018; pp. 172–188.
37. Hemon, A.; Lyonnet, B.; Rowe, F.; Fitzgerald, B. From Agile to DevOps: Smart Skills and Collaborations. *Inf. Syst. Front.* **2020**, *22*, 927–945. [CrossRef]
38. Marrero, L.; Astudillo, H. DevOps-RAF: An assessment framework to measure DevOps readiness in software organizations. In Proceedings of the 2021 40th International Conference of the Chilean Computer Science Society (SCCC), La Serena, Chile, 15–19 November 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8.
39. Pan, W.; Wei, H. Research on Key Performance Indicator (KPI) of Business Process. In Proceedings of the 2012 Second International Conference on Business Computing and Global Informatization, Shanghai, China, 12–14 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 151–154.
40. Durga Prasad, N.V.P.R.; Radhakrishna, C. New Key Performance Indicators (KPI) For Substation Maintenance Performance. In Proceedings of the 2019 International Conference on Computing, Power and Communication Technologies (GUCON), New Delhi, India, 27–28 September 2019; IEEE: Piscataway, NJ, USA, 2019. ISBN 9789353510985.
41. Zhu, H.; Bayley, I. If Docker is the Answer, What is the Question? In Proceedings of the 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), Bamberg, Germany, 26–29 March 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 152–163.
42. Weyuker, E.J.; Avritzer, A. A metric to predict software scalability. In Proceedings of the Eighth IEEE Symposium on Software Metrics, Ottawa, ON, Canada, 4–7 June 2002; IEEE Computer Society: Washington, DC, USA; pp. 152–158.
43. Dearle, A. Software Deployment, Past, Present and Future. In Proceedings of the Future of Software Engineering (FOSE '07), Minneapolis, MN, USA, 23–25 May 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 269–284.

44. Shahin, M.; Babar, M.A.; Zahedi, M.; Zhu, L. Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. In Proceedings of the 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Oulu, Finland, 11–12 October 2018; IEEE: Piscataway, NJ, USA, 2017; pp. 111–120.
45. Unger-Windeler, C.; Klunder, J.; Schneider, K. A Mapping Study on Product Owners in Industry: Identifying Future Research Directions. In Proceedings of the 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP), Montreal, QC, Canada, 25 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 135–144.
46. Tsunoda, M.; Matsumura, T.; Matsumoto, K. Modeling Software Project Monitoring with Stakeholders. In Proceedings of the 2010 IEEE/ACIS 9th International Conference on Computer and Information Science, Kaminoyama, Japan, 18–20 August 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 723–728.
47. Riungu-Kalliosaari, L.; Mäkinen, S.; Lwakatare, L.E.; Tiihonen, J.; Männistö, T. DevOps adoption benefits and challenges in practice: A case study. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Cham, Switzerland, 2016; Volume 10027 LNCS.
48. Iden, J.; Tessem, B.; Päivärinta, T. Problems in the interplay of development and IT operations in system development projects: A Delphi study of Norwegian IT experts. *Inf. Softw. Technol.* **2011**, *53*, 394–406. [CrossRef]
49. Tessem, B.; Iden, J. Cooperation between developers and operations in software engineering projects. In Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering, Leipzig, Germany, 13 May 2008; ACM: New York, NY, USA, 2008; pp. 105–108.
50. Woods, E. Aligning Architecture Work with Agile Teams. *IEEE Softw.* **2015**, *32*, 24–26. [CrossRef]
51. Govil, N.; Saurakhia, M.; Agnihotri, P.; Shukla, S.; Agarwal, S. Analyzing the Behaviour of Applying Agile Methodologies & DevOps Culture in e-Commerce Web Application. In Proceedings of the 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184), Tirunelveli, India, 15–17 June 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 899–902.
52. Khmelevsky, Y.; Li, X.; Madnick, S. Software development using agile and scrum in distributed teams. In Proceedings of the 2017 Annual IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 24–27 April 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–4.
53. Ebert, C.; Gallardo, G.; Hernantes, J.; Serrano, N. DevOps. *IEEE Softw.* **2016**, *33*, 94–100. [CrossRef]
54. Agrawal, P.; Rawat, N. Devops, A New Approach To Cloud Development & Testing. In Proceedings of the 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India, 27–28 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–4.
55. Choudhary, B.; Rakesh, S.K. An approach using agile method for software development. In Proceedings of the 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), Greater Noida, India, 3–5 February 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 155–158.
56. Dyba, T.; Dingsoyr, T. What Do We Know about Agile Software Development? *IEEE Softw.* **2009**, *26*, 6–9. [CrossRef]
57. Jabbari, R.; Bin Ali, N.; Petersen, K.; Tanveer, B. What is DevOps? In Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh, UK, 24 May 2016; ACM: New York, NY, USA, 2016; pp. 1–11.
58. Gokarna, M.; Singh, R. DevOps: A Historical Review and Future Works. In Proceedings of the 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida, India, 19–20 February 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 366–371.
59. Sharma, S.; Kumar, D.; Fayad, M.E. An Impact Assessment of Agile Ceremonies on Sprint Velocity Under Agile Software Development. In Proceedings of the 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 4–5 June 2020; IEEE: Piscataway, NJ, USA, 2021; pp. 1–5.
60. Srivastava, A.; Bhardwaj, S.; Saraswat, S. SCRUM model for agile methodology. In Proceedings of the 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, 5–6 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 864–869.
61. Paasivaara, M.; Durasiewicz, S.; Lassenius, C. Distributed Agile Development: Using Scrum in a Large Project. In Proceedings of the 2008 IEEE International Conference on Global Software Engineering, Bangalore, India, 17–20 August 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 87–95.
62. John, M.M.; Olsson, H.H.; Bosch, J. Towards MLOps: A Framework and Maturity Model. In Proceedings of the 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy, 1–3 September 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 334–341. [CrossRef]
63. Kang, H.; Le, M.; Tao, S. Container and microservice driven design for cloud infrastructure DevOps. In Proceedings of the 2016 IEEE International Conference on Cloud Engineering, IC2E 2016: Co-located with the 1st IEEE International Conference on Internet-of-Things Design and Implementation, Berlin, Germany, 4–8 April 2016.
64. Kubeflow–ML Toolkit. Available online: https://www.kubeflow.org/ (accessed on 26 August 2022).
65. Zhou, Y.; Yu, Y.; Ding, B. Towards MLOps: A Case Study of ML Pipeline Platform. In Proceedings of the 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), Beijing, China, 23–25 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 494–500.
66. Karamitsos, I.; Albarhami, S.; Apostolopoulos, C. Applying DevOps Practices of Continuous Automation for Machine Learning. *Information* **2020**, *11*, 363. [CrossRef]

67. Banerjee, A.; Chen, C.C.; Hung, C.C.; Huang, X.; Wang, Y.; Chevesaran, R. Challenges and experiences with MLOps for performance diagnostics in hybrid-cloud enterprise software deployments. In Proceedings of the OpML 2020—2020 USENIX Conference on Operational Machine Learning, Online, 28 July–7 August 2020.
68. Tamburri, D.A. Sustainable MLOps: Trends and Challenges. In Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, Timisoara, Romania, 1–4 September 2020.
69. Raj, E.; Buffoni, D.; Westerlund, M.; Ahola, K. Edge MLOps: An Automation Framework for AIoT Applications. In Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E), San Francisco, CA, USA, 4–8 October 2021; pp. 191–200. [CrossRef]
70. Granlund, T.; Kopponen, A.; Stirbu, V.; Myllyaho, L.; Mikkonen, T. MLOps challenges in multi-organization setup: Experiences from two real-world cases. In Proceedings of the 2021 IEEE/ACM 1st Workshop on AI Engineering—Software Engineering for AI, WAIN 2021, Madrid, Spain, 22–30 May 2021.
71. Azure MLOps. Available online: https://azure.microsoft.com/en-us/services/machine-learning/mlops/ (accessed on 26 August 2022).
72. AWS MLOps—Sagemaker. Available online: https://aws.amazon.com/sagemaker/mlops/ (accessed on 26 August 2022).
73. Practitioners Guide to MLOps—Google Cloud. Available online: https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf (accessed on 26 August 2022).
74. Dyck, A.; Penners, R.; Lichter, H. Towards Definitions for Release Engineering and DevOps. In Proceedings of the 2015 IEEE/ACM 3rd International Workshop on Release Engineering, Florence, Italy, 19 May 2015; IEEE: Piscataway, NJ, USA, 2015; p. 3.
75. What is Devops. Available online: https://www.atlassian.com/devops (accessed on 26 August 2022).
76. Cois, C.A.; Yankel, J.; Connell, A. Modern DevOps: Optimizing software development through effective system interactions. In Proceedings of the 2014 IEEE International Professional Communication Conference (IPCC), Pittsburgh, PA, USA, 13–15 October 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 1–7.
77. Virmani, M. Understanding DevOps & bridging the gap from continuous integration to continuous delivery. In Proceedings of the Fifth International Conference on the Innovative Computing Technology (INTECH 2015), Galicia, Spain, 20–22 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 78–82.
78. Kerzazi, N.; Adams MCIS, B.; Montreal, P. Who Needs Release and DevOps Engineers, and Why? In Proceedings of the International Workshop on Continuous Software Evolution and Delivery, Austin, TX, USA, 14–15 May 2016. [CrossRef]
79. Bankar, S.; Shah, D. Blockchain based framework for Software Development using DevOps. In Proceedings of the 2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), Navi Mumbai, India, 15–16 January 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
80. Soni, M. End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery. In Proceedings of the 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India, 25–27 November 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 85–89.
81. Alnafessah, A.; Gias, A.U.; Wang, R.; Zhu, L.; Casale, G.; Filieri, A. Quality-Aware DevOps Research: Where Do We Stand? *IEEE Access* **2021**, *9*, 44476–44489. [CrossRef]
82. Dalpiaz, F.; Brinkkemper, S. Agile Requirements Engineering with User Stories. In Proceedings of the 2018 IEEE 26th International Requirements Engineering Conference (RE), Banff, AL, Canada, 20–24 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 506–507.
83. Sarkan, H.M.; Ahmad, T.P.S.; Bakar, A.A. Using JIRA and Redmine in requirement development for agile methodology. In Proceedings of the 2011 Malaysian Conference in Software Engineering, Kuantan, Malaysia, 27–29 June 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 408–413.
84. Perera, P.; Silva, R.; Perera, I. Improve software quality through practicing DevOps. In Proceedings of the 2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, 6–9 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
85. Majumdar, R.; Jain, R.; Barthwal, S.; Choudhary, C. Source code management using version control system. In Proceedings of the 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 20–22 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 278–281.
86. Ren, Y.; Xing, T.; Quan, Q.; Zhao, Y. Software Configuration Management of Version Control Study Based on Baseline. In Proceedings of the 2010 3rd International Conference on Information Management, Innovation Management and Industrial Engineering, Kunming, China, 26–28 November 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 118–121.
87. Shah, J.; Dubaria, D.; Widhalm, J. A Survey of DevOps tools for Networking. In Proceedings of the 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 8–10 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 185–188.
88. Spinellis, D. Git. *IEEE Softw.* **2012**, *29*, 100–101. [CrossRef]
89. Hinsen, K.; Läufer, K.; Thiruvathukal, G.K. Essential Tools: Version Control Systems. *Comput. Sci. Eng.* **2009**, *11*, 84–91. [CrossRef]
90. Paez, N. Versioning Strategy for DevOps Implementations. In Proceedings of the 2018 Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI), Buenos Aires, Argentina, 28–30 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.

91. Palihawadana, S.; Wijeweera, C.H.; Sanjitha, M.G.T.N.; Liyanage, V.K.; Perera, I.; Meedeniya, D.A. Tool support for traceability management of software artefacts with DevOps practices. In Proceedings of the 2017 Moratuwa Engineering Research Conference (MERCon), Moratuwa, Sri Lanka, 29–31 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 129–134.
92. Git branching Model. Available online: https://nvie.com/posts/a-successful-git-branching-model/ (accessed on 26 August 2022).
93. Chang, C.-Y.; Ou, P.-P.; Deng, D.-J. Cross-Site Large-Scale Software Delivery with Enhanced Git Branch Model. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 18–20 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 153–156.
94. Louridas, P. Version control. *IEEE Softw.* **2006**, *23*, 104–107. [CrossRef]
95. Branching vs Trunk Based Development. Available online: https://launchdarkly.com/blog/git-branching-strategies-vs-trunk-based-development/ (accessed on 26 August 2022).
96. Spinellis, D. Package Management Systems. *IEEE Softw.* **2012**, *29*, 84–86. [CrossRef]
97. Sadowski, C.; Aftandilian, E.; Eagle, A.; Miller-Cushon, L.; Jaspan, C. Lessons from building static analysis tools at Google. *Commun. ACM* **2018**, *61*, 58–66. [CrossRef]
98. Jebbar, O.; Saied, M.A.; Khendek, F.; Toeroe, M. Poster: Re-Testing Configured Instances in the Production Environment—A Method for Reducing the Test Suite. In Proceedings of the 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), Xi'an, China, 22–27 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 367–370.
99. Chen, W.; Ye, K.; Wang, Y.; Xu, G.; Xu, C.-Z. How Does the Workload Look Like in Production Cloud? Analysis and Clustering of Workloads on Alibaba Cluster Trace. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 102–109.
100. Neitsch, A.; Wong, K.; Godfrey, M.W. Build system issues in multilanguage software. In Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 23–28 September 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 140–149.
101. Leite, L.; Rocha, C.; Kon, F.; Milojicic, D.; Meirelles, P. A Survey of DevOps Concepts and Challenges. *ACM Comput. Surv.* **2020**, *52*, 1–35. [CrossRef]
102. Pietrantuono, R.; Bertolino, A.; De Angelis, G.; Miranda, B.; Russo, S. Towards Continuous Software Reliability Testing in DevOps. In Proceedings of the 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), Montreal, QC, Canada, 27 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 21–27.
103. Hellmann, T.D.; Hosseini-Khayat, A.; Maurer, F. Supporting Test-Driven Development of Graphical User Interfaces Using Agile Interaction Design. In Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, Paris, France, 7–9 April 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 444–447.
104. Gohil, K.; Alapati, N.; Joglekar, S. Towards behavior driven operations (BDOps). In Proceedings of the 3rd International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2011), Bangalore, India, 14–15 September 2011; IET: London, UK, 2011; pp. 262–264.
105. Chen, B.; Song, J.; Xu, P.; Hu, X.; Jiang, Z.M. An automated approach to estimating code coverage measures via execution logs. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; ACM: New York, NY, USA, 2018; pp. 305–316.
106. Azure DevOps Code Coverage. Available online: https://docs.microsoft.com/fi-fi/azure/devops/pipelines/test/review-code-coverage-results?view=azure-devops (accessed on 26 August 2022).
107. Callanan, M.; Spillane, A. DevOps: Making It Easy to Do the Right Thing. *IEEE Softw.* **2016**, *33*, 53–59. [CrossRef]
108. Humble, J.; Read, C.; North, D. The Deployment Production Line. In Proceedings of the AGILE 2006 (AGILE'06), Minneapolis, MN, USA, 23–28 July 2006; IEEE: Piscataway, NJ, USA; pp. 113–118.
109. Lin, C.-C.; Wu, J.-J.; Lin, J.-A.; Song, L.-C.; Liu, P. Automatic Resource Scaling Based on Application Service Requirements. In Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 941–942.
110. Netto, M.A.S.; Cardonha, C.; Cunha, R.L.F.; Assuncao, M.D. Evaluating Auto-scaling Strategies for Cloud Computing Environments. In Proceedings of the 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, Paris, France, 9–12 September 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 187–196.
111. Anand, M. Cloud Monitor: Monitoring Applications in Cloud. In Proceedings of the 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India, 11–12 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–4.
112. Düllmann, T.F.; Paule, C.; van Hoorn, A. Exploiting DevOps Practices for Dependable and Secure Continuous Delivery Pipelines. In Proceedings of the 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE), Gothenburg, Sweden, 29 May 2018. [CrossRef]
113. Meyer, M. Continuous Integration and Its Tools. *IEEE Softw.* **2014**, *31*, 14–16. [CrossRef]
114. Vassallo, C.; Palomba, F.; Bacchelli, A.; Gall, H.C. Continuous code quality: Are we (really) doing that? In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; ACM: New York, NY, USA, 2018; pp. 790–795.
115. Owhadi-Kareshk, M.; Nadi, S.; Rubin, J. Predicting Merge Conflicts in Collaborative Software Development. In Proceedings of the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Recife, Brazil, 19–20 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–11.

116. AlAbwaini, N.; Aldaaje, A.; Jaber, T.; Abdallah, M.; Tamimi, A. Using Program Slicing to Detect the Dead Code. In Proceedings of the 2018 8th International Conference on Computer Science and Information Technology (CSIT), Amman, Jordan, 11–12 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 230–233.
117. Wang, X.; Zhang, Y.; Zhao, L.; Chen, X. Dead Code Detection Method Based on Program Slicing. In Proceedings of the 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Nanjing, China, 12–14 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 155–158.
118. Romano, S. Dead Code. In Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 23–29 September 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 737–742.
119. Abbass, M.K.A.; Osman, R.I.E.; Mohammed, A.M.H.; Alshaikh, M.W.A. Adopting Continuous Integeration and Continuous Delivery for Small Teams. In Proceedings of the 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), Khartoum, Sudan, 21–23 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–4.
120. Agarwal, A.; Gupta, S.; Choudhury, T. Continuous and Integrated Software Development using DevOps. In Proceedings of the 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), Paris, France, 22–23 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 290–293.
121. Runeson, P. A survey of unit testing practices. *IEEE Softw.* **2006**, *23*, 22–29. [CrossRef]
122. Grechanik, M.; Xie, Q.; Fu, C. Maintaining and evolving GUI-directed test scripts. In Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, Vancouver, BC, Canada, 16–24 May 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 408–418.
123. Misra, R.B. On determining the software testing cost to assure desired field reliability. In Proceedings of the IEEE INDICON 2004, First India Annual Conference, Kharagpur, India, 20–22 December 2004; IEEE: Piscataway, NJ, USA, 2004; pp. 517–520.
124. Komargodski, I.; Naor, M.; Yogev, E. White-Box vs. Black-Box Complexity of Search Problems: Ramsey and Graph Property Testing. In Proceedings of the 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), Berkeley, CA, USA, 15–17 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 622–632.
125. Horváth, F.; Bognár, S.; Gergely, T.; Rácz, R.; Beszédes, Á.; Marinković, V. Code Coverage Measurement Framework for Android Devices. *Acta Cybern.* **2014**, *21*, 439–458. [CrossRef]
126. Adams, B.; McIntosh, S. Modern Release Engineering in a Nutshell—Why Researchers Should Care. In Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 14–18 March 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 78–90.
127. Shahin, M.; Ali Babar, M.; Zhu, L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* **2017**, *5*, 3909–3943. [CrossRef]
128. Build Artifact. Available online: https://www.jetbrains.com/help/teamcity/build-artifact.html (accessed on 26 August 2022).
129. Storing Build Artifact. Available online: https://circleci.com/docs/2.0/artifacts/ (accessed on 26 August 2022).
130. Rufino, J.; Alam, M.; Ferreira, J. Monitoring V2X applications using DevOps and docker. In Proceedings of the 2017 International Smart Cities Conference (ISC2), Wuxi, China, 14–17 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.
131. Symeonidis, G.; Nerantzis, E.; Kazakis, A.; Papakostas, G.A. MLOps-Definitions, Tools and Challenges. In Proceedings of the 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 26–29 January 2022.
132. Iterative Incremental Process. Available online: https://ml-ops.org/content/mlops-principles (accessed on 26 August 2022).
133. CD4ML—Continuous Delivery for Machine Learning. Available online: https://martinfowler.com/articles/cd4ml.html (accessed on 14 December 2021).
134. Studer, S.; Bui, T.B.; Drescher, C.; Hanuschkin, A.; Winkler, L.; Peters, S.; Mueller, K.-R. Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. *Mach. Learn. Knowl. Extr.* **2021**, *3*, 392–413. [CrossRef]
135. Santa, R.; Bretherton, P.; Ferrer, M.; Soosa, C.; Hyland, P. The role of cross-functional teams on the alignment between technology innovation effectiveness and operational effectiveness. *Int. J. Technol. Manag.* **2011**, *55*, 122–137. [CrossRef]
136. Alahmari, S.S.; Goldgof, D.B.; Mouton, P.R.; Hall, L.O. Challenges for the Repeatability of Deep Learning Models. *IEEE Access* **2020**, *8*, 211860–211868. [CrossRef]
137. Victor, K.F.; Michael, I.Z. Intelligent data analysis and machine learning: Are they really equivalent Concepts? In Proceedings of the 2017 Second Russia and Pacific Conference on Computer Technology and Applications (RPC), Vladivostok, Russia, 25–29 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 59–63.
138. Hafen, R.; Critchlow, T. EDA and ML—A Perfect Pair for Large-Scale Data Analysis. In Proceedings of the 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 20 May 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 1894–1898.
139. Guruvayur, S.R.; Suchithra, R. A detailed study on machine learning techniques for data mining. In Proceedings of the 2017 International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, India, 11–12 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1187–1192.
140. 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. [CrossRef]
141. Wang, L.; Han, M.; Li, X.; Zhang, N.; Cheng, H. Review of Classification Methods on Unbalanced Data Sets. *IEEE Access* **2021**, *9*, 64606–64628. [CrossRef]

142. Lwakatare, L.E.; Range, E.; Crnkovic, I.; Bosch, J. On the Experiences of Adopting Automated Data Validation in an Industrial Machine Learning Project. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Madrid, Spain, 25–28 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 248–257.
143. Subramanya, R.; Yli-Ojanperä, M.; Sierla, S.; Hölttä, T.; Valtakari, J.; Vyatkin, V. A Virtual Power Plant Solution for Aggregating Photovoltaic Systems and Other Distributed Energy Resources for Northern European Primary Frequency Reserves. *Energies* **2021**, *14*, 1242. [CrossRef]
144. Galhotra, S.; Khurana, U.; Hassanzadeh, O.; Srinivas, K.; Samulowitz, H.; Qi, M. Automated Feature Enhancement for Predictive Modeling using External Knowledge. In Proceedings of the 2019 International Conference on Data Mining Workshops (ICDMW), Beijing, China, 8–11 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1094–1097.
145. Pham, M.; Knoblock, C.A.; Pujara, J. Learning Data Transformations with Minimal User Effort. In Proceedings of the 2019 International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 657–664.
146. Marculescu, D.; Stamoulis, D.; Cai, E. Hardware-aware machine learning. In Proceedings of the International Conference on Computer-Aided Design, San Diego, CA, USA, 5–8 November 2018; ACM: New York, NY, USA, 2018; pp. 1–8.
147. Gada, M.; Haria, Z.; Mankad, A.; Damania, K.; Sankhe, S. Automated Feature Engineering and Hyperparameter optimization for Machine Learning. In Proceedings of the 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 19–20 March 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 981–986.
148. Kahloot, K.M.; Ekler, P. Algorithmic Splitting: A Method for Dataset Preparation. *IEEE Access* **2021**, *9*, 125229–125237. [CrossRef]
149. Medar, R.; Rajpurohit, V.S.; Rashmi, B. Impact of Training and Testing Data Splits on Accuracy of Time Series Forecasting in Machine Learning. In Proceedings of the 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, India, 17–18 August 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
150. Li, S.; Hu, W.; Cao, D.; Dragičević, T.; Huang, Q.; Chen, Z.; Blaabjerg, F. Electric Vehicle Charging Management Based on Deep Reinforcement Learning. *J. Mod. Power Syst. Clean Energy* **2021**, *10*, 719–730. [CrossRef]
151. Posoldova, A. Machine Learning Pipelines: From Research to Production. *IEEE Potentials* **2020**, *39*, 38–42. [CrossRef]
152. Marijan, D.; Gotlieb, A.; Kumar Ahuja, M. Challenges of Testing Machine Learning Based Systems. In Proceedings of the 2019 International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, 4–9 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 101–102.
153. Gisselaire, L.; Cario, F.; Guerre-berthelot, Q.; Zigmann, B.; du Bousquet, L.; Nakamura, M. Toward Evaluation of Deployment Architecture of ML-Based Cyber-Physical Systems. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), San Diego, CA, USA, 11–15 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 90–93.
154. Barque, M.; Martin, S.; Vianin, J.E.N.; Genoud, D.; Wannier, D. Improving wind power prediction with retraining machine learning algorithms. In Proceedings of the 2018 International Workshop on Big Data and Information Security (IWBIS), Jakarta, Indonesia, 12–13 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 43–48.
155. Fields, T.; Hsieh, G.; Chenou, J. Mitigating Drift in Time Series Data with Noise Augmentation. In Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 5–7 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 227–230.
156. Bock, S.; Weis, M. A Proof of Local Convergence for the Adam Optimizer. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–8.
157. Wang, K.; Gopaluni, R.B.; Chen, J.; Song, Z. Deep Learning of Complex Batch Process Data and Its Application on Quality Prediction. *IEEE Trans. Ind. Inform.* **2020**, *16*, 7233–7242. [CrossRef]
158. Qian, C.; Yu, W.; Liu, X.; Griffith, D.; Golmie, N. Towards Online Continuous Reinforcement Learning on Industrial Internet of Things. In Proceedings of the 2021 SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI), Atlanta, GA, USA, 18–21 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 280–287.
159. Feature Store Comparision. Available online: https://mlops.community/learn/feature-store/ (accessed on 26 August 2022).
160. Metadata Management MLOps. Available online: https://mlops.community/learn/metadata-storage-and-management/ (accessed on 26 August 2022).
161. MLOps Pipelines in ML. Available online: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning (accessed on 26 August 2022).
162. de la Nieta, A.A.S.; Gibescu, M. Day-ahead Scheduling in a Local Electricity Market. In Proceedings of the 2019 International Conference on Smart Energy Systems and Technologies (SEST), Porto, Portugal, 9–11 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6.
163. Zhao, Q.; Shen, Y.; Li, M. Control and Bidding Strategy for Virtual Power Plants With Renewable Generation and Inelastic Demand in Electricity Markets. *IEEE USA Trans. Sustain. Energy* **2016**, *7*, 562–575. [CrossRef]
164. Giovanelli, C.; Sierla, S.; Ichise, R.; Vyatkin, V. Exploiting artificial neural networks for the prediction of ancillary energy market prices. *Energies* **2018**, *11*, 1906. [CrossRef]
165. Sierla, S.; Pourakbari-Kasmaei, M.; Vyatkin, V. A taxonomy of machine learning applications for virtual power plants and home/building energy management systems. *Autom. Constr.* **2022**, *136*, 104174. [CrossRef]

166. Alanne, K.; Sierla, S. An overview of machine learning applications for smart buildings. *Sustain. Cities Soc.* **2022**, *76*, 103445. [CrossRef]
167. Sierla, S.; Ihasalo, H.; Vyatkin, V. A Review of Reinforcement Learning Applications to Control of Heating, Ventilation and Air Conditioning Systems. *Energies* **2022**, *15*, 3526. [CrossRef]
168. Subramanya, R.; Sierla, S.A.; Vyatkin, V. Exploiting Battery Storages With Reinforcement Learning: A Review for Energy Professionals. *IEEE Access* **2022**, *10*, 54484–54506. [CrossRef]
169. Chouhan, B.S.; Rao, K.V.S.; Kumar Saxena, B. Reduction in carbon dioxide emissions due to wind power generation in India. In Proceedings of the 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon), Bengaluru, India, 17–19 August 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 257–264.
170. Takano, H.; Zhang, P.; Murata, J.; Hashiguchi, T.; Goda, T.; Iizaka, T.; Nakanishi, Y. A Determination Method for the Optimal Operation of Controllable Generators in Micro Grids That Copes with Unstable Outputs of Renewable Energy Generation. *Electr. Eng. Jpn.* **2015**, *190*, 56–65. [CrossRef]
171. Karhula, N.; Sierla, S.; Vyatkin, V. Validating the Real-Time Performance of Distributed Energy Resources Participating on Primary Frequency Reserves. *Energies* **2021**, *14*, 6914. [CrossRef]
172. Aaltonen, H.; Sierla, S.; Subramanya, R.; Vyatkin, V. A Simulation Environment for Training a Reinforcement Learning Agent Trading a Battery Storage. *Energies* **2021**, *14*, 5587. [CrossRef]
173. Weron, R. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *Int. J. Forecast.* **2014**, *30*, 1030–1081. [CrossRef]
174. Saâdaoui, F. A seasonal feedforward neural network to forecast electricity prices. *Neural Comput. Appl.* **2017**, *28*, 835–847. [CrossRef]
175. Lim, B.; Arik, S.O.; Loeff, N.; Pfister, T. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *Int. J. Forecast.* **2019**, *37*, 1748–1764. [CrossRef]