Bridging the Gap: Empowering Machine Learning Development with Service-Oriented Computing Principles

Mostafa Hadadian Nejad Yousefi[1], Viktoriya Degeler[2], and Alexander Lazovik[1]

[1] University of Groningen, Faculty of Science and Engineering, Bernoulli Institute
{m.hadadian,a.lazovik}@rug.nl
[2] University of Amsterdam, Faculty of Science, Informatics Institute
v.o.degeler@uva.nl

Abstract. The software industry has been utilizing services to achieve a high level of maturity in software development for several years now. However, machine learning development, despite its rapidly growing popularity, is still lagging behind in terms of adopting services-oriented computing practices. The reasons behind this gap are multifaceted, ranging from the unique challenges of machine learning development to the lack of a cohesive framework for integrating services into the development process. In this paper, we aim to shed light on the disparities between services-oriented computing and machine learning development, and present our vision for bridging this gap. We propose a set of additional steps that need to be taken to empower machine learning development using services-oriented computing. We proposed an architecture that seamlessly integrates our solutions, enabling users to efficiently manage and orchestrate complex machine learning systems. By leveraging the best practices of services-oriented computing, we believe that machine learning development can achieve a higher level of maturity, improve the efficiency of the development process, and ultimately, facilitate the more effective creation of machine learning applications.

Keywords: Machine Learning Lifecycle · MLOps · Service-Oriented Computing · Adaptive Data Processing · ML Pipelines

1 Introduction

Machine learning (ML) has emerged as a powerful tool for solving complex problems across various domains, leading to a growing demand for production-grade ML applications. With the increasing importance of ML in various industries, the need for efficient and scalable ML development has become more pronounced [37]. However, despite the rapid advancements in ML techniques and tools, the development of production-grade ML systems still faces several challenges that hinder its alignment with best practices in software development [64].

In recent years, the software industry has successfully embraced service-oriented computing, which has brought about significant improvements in soft-

ware development processes, enabling better modularity, flexibility, and maintainability [51]. However, ML development has not yet fully adopted these best practices, resulting in a gap between service-oriented computing and ML development [3].

In this paper, we investigate the best practices in service-oriented computing and software development and identify the gaps between these practices and ML development. We propose a comprehensive solution for bridging these gaps and present an architecture to integrate all the proposed solutions, based on the "Everything as a Module" (XaaM) vision. XaaM serves as an extension to the "Everything as a Service" (XaaS) paradigm, specifically introduced to differentiate between conventional web services and machine learning services, while also catering to the machine learning community's preference for the term "Module." Our work aims to empower ML development by incorporating the best practices of service-oriented computing, ultimately leading to more mature, efficient, and scalable ML systems.

The benefits of our work include improved efficiency in the development process, increased scalability and adaptability of ML applications, and more effective creation of production-grade ML systems. By developing and refining our vision, we contribute to the ongoing effort to enhance ML development practices and the broader adoption of service-oriented computing in the ML domain.

The paper is structured as follows: In Section 2, we provide the necessary background to enhance the readers' understanding of the machine learning development lifecycle. Section 3 presents a review of related work. Section 4 identifies the gaps that exist in various aspects and proposes solutions to bridge them. This section concludes with the introduction of an architecture for integrating the proposed solutions. Lastly, in Section 5, we present the conclusions of the paper.

2 Background

In this section, we discuss the lifecycle of Artificial Intelligence/Machine Learning (AI/ML) applications. Drawing from a diverse range of literature [1,32,61,66,68,73], we synthesize the common elements into a generalized workflow depicted in Figure 1.

We provide an overview of the various stages involved, noting that specific details may differ across implementations. The latter three stages are emphasized, as they are of particular relevance to the scope of this work.

The AI/ML lifecycle encompasses five interrelated stages.

First, during the *business requirement* stage, stakeholders collaborate with the AI/ML team to define the problem, objectives, and project scope. Second, the data preparation stage entails acquiring, cleaning, preprocessing, and transforming data for model training and evaluation. Third, the AI/ML development stage focuses on designing, implementing, and validating machine learning models. Fourth, the application deployment stage involves integrating developed models

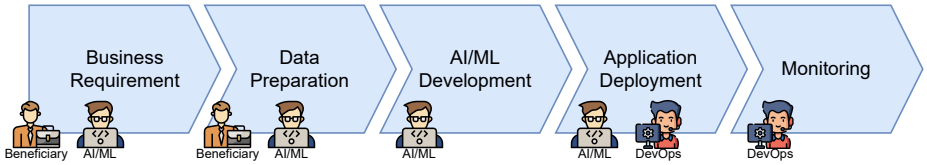

Fig. 1. Coarse-grained AI/ML application life-cycle illustrating the stages and their corresponding actors.

into a functional application and ensuring its stability, performance, and security in a production environment. Finally, the monitoring stage involves tracking the application's performance, identifying issues, and gathering insights for continuous improvement. Each stage presents unique challenges and necessitates the expertise of various team members to deliver a successful, high-quality AI/ML solution.

Data preparation steps differ significantly between the AI/ML lifecycle and traditional software engineering, in particular due to iteration frequency. It happens quite often to revisit earlier stages in the process. For example, during ML model development and testing, it might become apparent that the available data is insufficient to address the problem. Likewise, integration testing may reveal communication difficulties between ML components and other parts of the system due to data incompatibility, necessitating a return to the development stage. Additionally, monitoring might uncover performance degradation or operational issues, prompting revisiting the development and deployment stages. It is worth noting that each stage possesses an iterative nature as well. For instance, the development stage involves extensive iterations focused on model configuration and tuning.

AI/ML development diverges from traditional software development in plenty of ways. Firstly, it involves constructing probabilistic models that learn patterns from data, requiring a more iterative and experimental approach. This often results in AI/ML teams exploring various model architectures and algorithms before settling on the most effective solution. Secondly, the quality of an AI/ML solution is heavily reliant on the data used for training, thus emphasizing the significance of data preparation, including data acquisition, cleaning, and preprocessing. In contrast, classic software development places less emphasis on data quality. Furthermore, AI/ML development demands specialized skills and expertise in machine learning, statistics, and domain knowledge, whereas traditional software development focuses on programming, system design, and software engineering principles.

Evaluating a learning model is more challenging compared to classic software development, as model performance is highly dependent on the data. Consequently, AI/ML teams must conduct extensive training and testing on both small and large datasets that closely resemble production data, necessitating a scalable underlying platform to run the experiments efficiently. This iterative process ensures that the AI/ML solution generalizes well to new, unseen data

and effectively addresses the defined business requirements. Additionally, AI/ML development often involves uncertainty and requires more rigorous validation methods compared to traditional software development, where the correctness of an algorithm is often more deterministic.

The *deployment* stage of an application exhibits several distinctive characteristics when compared to traditional software deployment methodologies. Deploying AI/ML models often entails additional complexities, such handling the compute-intensive nature of ML models and as ensuring model reproducibility where running the same application with the same data should ideally yield the same output, despite the inherent probabilistic nature of ML models.

AI/ML applications are frequently developed by individuals who primarily focus on code and learning aspects rather than software engineering practices. This can make packaging the application, creating a data processing pipeline, and establishing proper communication between components challenging. Furthermore, rolling updates or rollbacks in AI/ML deployments involve more than just replacing a container, as they also require updating the model data. This adds an extra layer of complexity to the deployment process, necessitating close collaboration between the AI/ML team and the DevOps team to manage these challenges effectively. MLOps also emphasizes the importance of automating and streamlining the deployment process, as AI/ML models often require frequent updates and retraining to maintain their accuracy and relevance.

The process of *monitoring* in the AI/ML lifecycle showcases several unique features that distinguish it from conventional software development practices. In AI/ML lifecycle, the primary focus is on tracking model performance and accuracy, which may change over time due to factors such as data drift or model degradation. This requires specialized monitoring tools and techniques, such as data drift detection and concept drift analysis. Moreover, the iterative nature of AI/ML development necessitates continuous feedback loops and frequent model updates to maintain optimal performance. In contrast, classic software development monitoring is more focused on application performance, resource utilization, and uptime, with updates typically centered around bug fixes, feature enhancements, and addressing security vulnerabilities.

In summary, the AI/ML lifecycle presents distinct challenges and considerations compared to traditional software development. Key differences include the iterative and experimental nature of AI/ML development, the emphasis on data quality, and the need for specialized skills. Additionally, evaluating learning models, deployment, and monitoring in MLOps entail unique complexities. By understanding these fundamental divergences, organizations can better adapt their practices to successfully deliver high-quality AI/ML solutions that drive value and innovation across various domains.

3 Related Work

Research in the field of machine learning has witnessed a rapid surge in recent years, and consequently, a growing number of studies have addressed the

challenges of applying service-oriented computing principles to machine learning development. In this section, we briefly review the most relevant works in this area.

MLOps [32,61,68,73] is an emerging paradigm that aims to bridge the gap between machine learning development and traditional software development by incorporating DevOps principles. It focuses on automating and streamlining the machine learning development, deployment, and monitoring processes to increase efficiency and reduce the time to market. Several studies [1,66] have proposed MLOps frameworks and tools to facilitate this integration.

Another area of interest is the application of microservices architecture in machine learning development [18]. Microservices enable the modularization of machine learning components, allowing for more flexible development and deployment of machine learning applications. This approach also promotes the reusability of machine learning components, which can significantly reduce development time and effort.

In addition to these studies, research works [27,14] have investigated the integration of machine learning models into service-oriented architectures. These studies primarily focus on identifying the challenges and opportunities of using service-oriented computing principles in machine learning development, as well as proposing methods and architectures to support this integration.

While these works provide valuable insights into the application of service-oriented computing in machine learning development, they often focus on specific aspects, such as MLOps or microservices, rather than presenting a comprehensive vision for empowering machine learning development using service-oriented computing principles. In this paper, we aim to provide a broader perspective on this topic by discussing a wide range of techniques and principles that can be employed to bridge the gap between machine learning development and service-oriented computing.

4 Methods

In this section, we examine various aspects of software development and identify existing gaps, using Service Oriented Computing as a reference point. We begin by introducing modules and explaining module composition. Before delving into our vision for adaptive module selection, we discuss the essential requirements of version control and monitoring. We then address the management of machine learning application lifecycles and the integration of "what-if" scenarios for enhanced testing and interpretability. Subsequently, we describe the implementation of an underlying runtime controller to support the solutions presented in this section. Finally, we conclude with a layered software architecture that demonstrates the cohesive integration of these solutions.

4.1 Modularity By Design

Modularity by design refers to an approach that emphasizes the creation of smaller, independent, and interchangeable services. These services can be as-

sembled, rearranged, or replaced without affecting the overall system's functionality. The primary advantages of modular design include increased flexibility, reusability, maintainability, and scalability.

In our research, we distinguish between two levels of modularity within the context of machine learning applications: 1) Algorithmic Modularity and 2) Architectural Modularity. This classification highlights different aspects of machine learning applications, ranging from programming and code-level details to larger-scale system architecture and deployment considerations.

Algorithmic Modularity pertains to the utilization of programming languages or frameworks for the development of machine learning applications. Data scientists often employ frameworks such as Scikit-learn [53] or PyTorch [52] to facilitate various stages of their ML applications, including data preprocessing, scaling, modeling, and evaluation. Leveraging these frameworks enables the effective modularization of ML applications and accelerates the development process.

Architectural Modularity, on the other hand, involves packaging each stage into distinct services and deploying these services into appropriate environments, such as production. This level of modularity offers greater flexibility, improved maintainability, and enhanced scalability, ensuring that the ML application remains adaptable to changing requirements and emerging technological advancements.

Due to the wealth of available frameworks in machine learning application development, algorithmic modularity is well-established. AI / ML teams generally concentrate on the primary purpose of their applications and wish to avoid unnecessary complexities, such as packaging (e.g., containerization) or deployment [48]. They often create monolithic applications that may be deployed using services but remain monolithic by design, failing to exploit modularity's full potential.

As architectural modularity is less widespread, our primary goal is to promote its adoption and increase its prevalence in the field. Architectural modularity provides numerous benefits, including enhanced maintainability, scalability, and adaptability to changing requirements. By advocating for its adoption, we aim to facilitate the development of more robust and flexible machine learning applications.

In addition to the applicability, our proposed solution must be both user-friendly and easy to understand to ensure its acceptance within the community. Our approach aims to facilitate the seamless integration of services, promote efficient collaboration between different teams, and simplify the development process. To achieve this, we define modules as a higher abstraction of services with two main components rooted in the concept of polymorphism: Module Definition and Module Implementation. These components ensure applicability and enhance understandability by providing a clear separation between high-level and low-level information of a module.

Module Definition involves creating a general and unified interface for modules, similar to APIs for services. These interfaces facilitate communication

among team members and between different teams, ensuring everyone has a clear understanding of each module, e.g. what are its purpose, inputs, and outputs. The clarity of module definition enables a better division of responsibilities. For instance, a clear Module Definition allows AI/ML teams to focus on the internal logic of modules, while DevOps teams handle packaging and deployment.

Module Implementation refers to the process of putting a module definition into practice, much like how concrete classes in object-oriented programming languages implement abstract classes. This method permits multiple implementations of a single module definition, fostering reusability and polymorphism within the system. Importantly, a module implementation can be composed of multiple smaller modules.

Throughout the remainder of this paper, we will use the term "module" to refer to services in the context of our research. Specifically, we will focus on machine learning services that are designed, implemented, and maintained using MLOps best practices, as well as the solutions we propose. We selected the term "module" due to its common usage in the machine learning field, where it denotes a self-contained and coherent unit of work that shares similarities with the concept of a service.

In essence, any combination of a module definition and module implementation forms a module, as illustrated in Figure 3a. It is important to note that the definitions and implementations of the modules are loosely coupled. If a module implementation fulfills a module definition, they can be combined to create a module. Consequently, a single module definition may be satisfied by multiple implementations, and a module implementation may satisfy more than one definition. This flexibility is a significant advantage of our approach.

Modules that share the same module definition are considered equivalent, as they achieve the same objective using distinct methodologies. However, it is essential to acknowledge that equivalent modules may display different performances when handling the same tasks due to the variability in their implementations.

To illustrate, let us consider an example of a module definition and two module implementations. Imagine a module definition called "Scaler" where the input is a matrix of numeric data with shape (`n-samples`, `n-features`), and the output is a standardized version of this matrix with the same shape. The first implementation, "StandardScaler," standardizes the input features to have zero mean and unit variance by subtracting the mean and dividing by the standard deviation of each feature. The second implementation, "MinMaxScaler" Rescales the input features to a specified range (usually [0, 1]) by subtracting the minimum value and dividing by the range of each feature. Although both implementations are scaling the input features, their output has a different distribution.

Adopting the XaaS [25] paradigm, we propose a new concept called "Everything as a Module" (XaaM), which presents a general unified interface that facilitates the encapsulation of diverse components in a machine learning system, including executable codes, ML pipelines, and datasets. Figure 2 offers a

demonstrative instance of XaaM for the training and inference stages of machine learning applications, where each artifact is considered a module.

Using various implementations, XaaM enhances the modularity and flexibility of AI/ML applications, ultimately advancing the state-of-the-art and advancing innovation in the field.

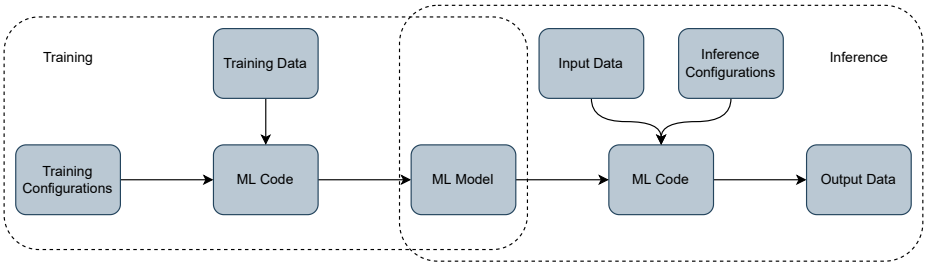

Fig. 2. An example XaaM demonstration for training and inference

4.2 Service Composition

Service composition is the process of combining simpler services to form a more complex one, enabling developers to break down intricate systems into manageable components. Our goal is to create modular and adaptable modules that can be easily adjusted and extended to meet evolving requirements. Similar to Web Service Composition [7], we define two module types:

- **Atomic Module**: A self-contained module independent of other modules, such as a Docker container.
- **Composite Module**: A module composed of multiple atomic or composite modules, like a data processing pipeline consisting of Scaler and Model modules, as shown in Figure 3c.

Our framework houses module definitions and implementations in a registry (Figure 3b). A composite module is represented by a graph topology, which details the included modules and their connections. Both atomic and composite modules share consistent definitions, enabling polymorphism and module reuse. Topology structures can adopt any form, unlike works that enforce sequential steps [53,72] or Directed Acyclic Graph (DAG) [11,45] structure.

To create a composite module adhering to a desired definition, developers outline constituent module definitions and connections, choose suitable implementations, and align the resulting composite implementation with the desired module definition. Architectural modularity allows seamless alteration of module implementations without code changes or complex procedures.

We can conduct operational and behavioral verifications with well-defined module structures. Operational verification confirms the pipeline's correctness,

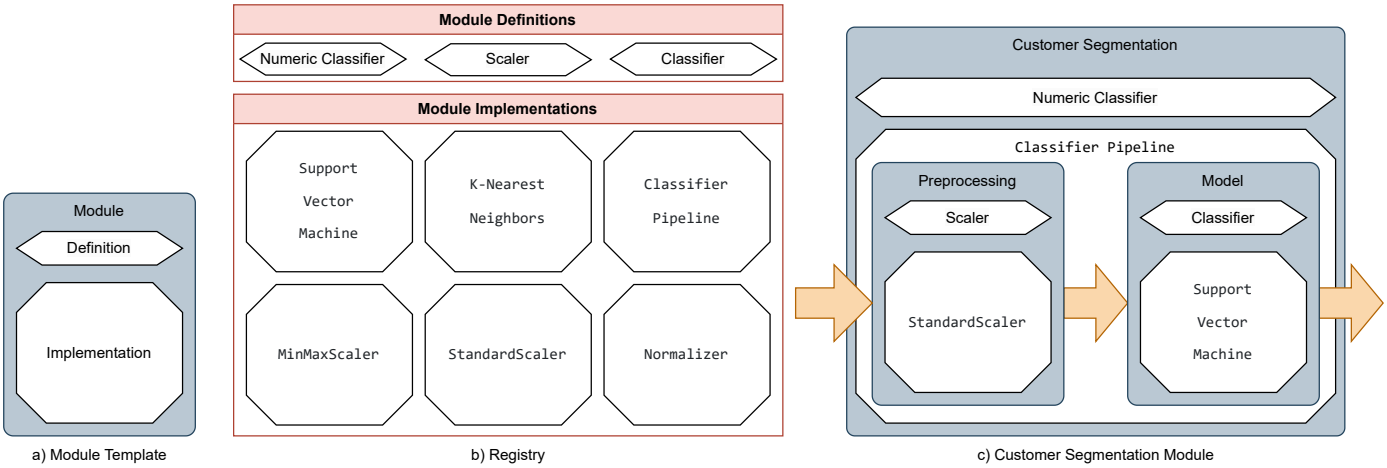

Fig. 3. Module Composition Example

while behavioral verification assesses whether the composite module produces the expected output. The former relies on module implementations, and the latter depends on module definitions.

We propose two automation stages for module composition: 1) Topology Creation and 2) Implementation Selection. The first stage generates a topology using modules from the registry or creating missing module definitions. The second stage selects appropriate implementations for each definition, ensuring that the chosen implementations can interact effectively and process data efficiently.

Module composition in machine learning development differs from service composition due to its probabilistic nature. Performance cannot be guaranteed through testing on available data alone, but we can use historical data and feedback to make informed decisions during module composition.

There are several techniques available for the automation of machine learning application creation, such as AutoAI [17] and AutoML [33]. While these approaches hold significant promise, they are not without their challenges, as identified by Elshawi et al. [26]. In our work, we aim to address several of these challenges, including Composability, Scalability, and Continuous delivery pipeline.

One of the primary challenges with existing AutoAI and AutoML approaches is their lack of generality and flexibility, particularly with respect to composability. These approaches often lack the ability to incorporate custom components or allow users to tweak the generated machine learning pipeline. Our approach aims to address this limitation by providing a fully automated solution that is still general enough to allow users to be involved in various formats. For instance, users can define certain parts of a composite module and let the system fill in the rest, allowing for greater flexibility and customizability.

In addition to addressing the issue of composability, we also seek to tackle challenges related to scalability, which can be especially problematic for large real datasets. To overcome this challenge, we plan to utilize techniques such as meta-learning to learn from previous runs and gradually improve the composite module's performance. This aligns with our goal of establishing a continuous delivery pipeline for machine learning applications, enabling us to deliver more effective solutions to end-users while improving overall efficiency and scalability.

In summary, our approach to module composition provides greater composability, flexibility, and scalability, allowing for more customized and adaptable machine learning pipelines. By leveraging historical data and feedback, we can make more informed decisions during module composition, which contributes to the continuous improvement of the composite module's performance. Ultimately, our approach allows for the development of more effective machine learning applications while reducing development time and costs.

4.3 Version Control

Version control is essential in both software engineering and machine learning, managing artifacts like source code, configuration files, and documentation in the former [74], and tracking changes in data, models, hyperparameters, and code in the latter [12,5]. Effective version control systems facilitate collaboration, reproducibility, and debugging.

Git has become the industry standard in version control for software engineering, with popular implementations like GitHub, GitLab, and BitBucket. It can manage common artifacts in machine learning, such as hyperparameters stored as plain text. However, managing large datasets and model weights presents unique challenges [34,35,69] due to factors like storage requirements and the diversity of machine learning frameworks.

To address these challenges, specialized version control systems such as DVC [36], Pachyderm [50], and MLflow [72] have been developed. They offer features like model versioning, data versioning, and model lineage tracking, simplifying the management of model and data updates. Cloud-based solutions like Amazon S3, Azure Blob Storage, and Google Cloud Storage can provide scalable storage and versioning solutions for large models and datasets.

In our XaaM vision, we treat data and other components as modules, ensuring consistency across various projects and teams. We store module definitions and implementations in an informative text format, enabling Git-based version control. This approach simplifies project management and ensures consistency, facilitating the integration of different modules and tools for developing complex and scalable machine learning solutions.

We can still utilize specialized tools like DVC, Pachyderm, and MLflow for versioning, but we present everything as a module from the users' and high-level system's perspective. We aim to maintain module lineage, improving transparency and reproducibility by enabling users to trace the history and evolution of each module in the project. For instance, users can determine the origin of the output data, including the model, configuration, and input data involved in

its generation, and trace the process through which an ML model was produced, including its training specifics.

In summary, our XaaM approach focuses on treating all components of the machine learning project as modules, ensuring consistency and manageability across different teams and projects. By leveraging both traditional version control systems like Git and specialized tools tailored for machine learning, we bridge the gap between machine learning development and software development, ultimately leading to more effective and streamlined machine learning projects with enhanced transparency and reproducibility through module lineage tracking.

4.4 Continuous Monitoring

Continuous monitoring is vital for managing application health and performance in software development projects, particularly in service-based applications with complex interdependencies [62,24,2]. In machine learning projects, continuous monitoring ensures model performance remains consistent, detects data drifts, anomalies, and performance issues, and determines when model retraining is necessary [68,60,38,66].

Challenges in continuous monitoring include integrating monitoring metrics and KPI evaluations from different teams, scalability and real-time monitoring, and addressing the statistical nature of drift detection and outlier identification [68,66,67,65,44,15,40,13,64]. Our XaaM vision addresses these challenges by building on state-of-the-art monitoring techniques that adapt to changing requirements and workloads without adversely affecting performance.

We propose **Monitoring as a Module** to integrate various monitoring techniques and enable seamless collaboration between teams. This approach standardizes data collection and storage methods [49], monitors upstream processes that feed data to ML systems [64], and uses input data statistics and output predictions as a proxy for model performance when labels are not readily available [13]. Furthermore, our vision emphasizes monitoring prediction bias, enforcing action limits to avoid unintended consequences, and leveraging module lineage for valuable insights into module impact and associations [64].

In summary, our continuous monitoring vision combines ML development and software development by introducing "Monitoring as a Module" to facilitate collaboration, standardization, and enhanced monitoring processes. By focusing on key aspects such as upstream process monitoring, prediction bias detection, action limit enforcement, and leveraging module lineage, we aim to develop comprehensive, scalable, and adaptive monitoring solutions.

4.5 Adaptive Service Selection

Module selection involves searching and identifying module implementations that align with a specific module definition and its requirements, resembling service composition in web services. In our vision, selection emphasizes choosing existing implementations, while composition focuses on generating new modules [47].

This process can occur at three stages in the module composition life cycle: design time, deployment time, and runtime [41].

During design time, developers create and define a module tailored to fulfill specific requirements. Deployment time involves installing and configuring the composition in the runtime environment for execution. Runtime is when the module is executed, and its performance and functionality are evaluated. Module selection at runtime depends on algorithms that associate module definitions with implementations based on performance metrics and requirements, ensuring the selection of the most suitable implementation [64].

Adaptive service selection algorithms prioritize QoS attributes such as response time, success rate, and cost [22,71]. In machine learning development, we must ensure QoS metrics while satisfying performance requirements, like accuracy and Mean Squared Error (MSE) [21,30]. Addressing the interdependence of metrics is crucial, considering modules correlations and user requirements correlations [42,55].

Our vision's module selection consists of three primary stages, each presenting unique challenges:

- **candidate selection:** identifying satisfying module implementations for a given module definition.
- **Ranking**: ranking the best-performing module implementations for a specific situation.
- **Choosing**: deciding whether it is worth updating the current module in production.

Challenges during the candidate selection phase include ensuring implementation satisfaction of the module definition and designing a scalable find-matching algorithm. The level of granularity poses another challenge, as a module implementation may be a composite module. Incorporating the structural-semantic approach enhances the candidate identification phase, using domain ontology concepts, similarity measures, and structural properties analysis to select suitable module implementations [28].

Defining the situation is a challenge in the ranking phase, which depends on incoming data, environments, and requirements. Deciding how much history to consider and predicting the future is crucial. After ranking, the choosing phase involves deciding whether to update the current module in production, considering the costs of redeployment and potential suboptimal performance measures.

Designing a reliable and fast adaptive module selection involves numerous interconnected choices across all phases. Understanding these choices' influence on each other and the overall system performance is essential. A holistic approach is necessary to create a system that can adapt effectively to changing situations and maintain optimal performance across various scenarios.

In conclusion, adaptive module selection is critical for ensuring machine learning systems' requirements. By addressing the unique challenges in each phase and considering the complexities and interdependencies of adaptive module selection,

we can develop a robust approach that adapts to ever-changing circumstances, delivering high-performing and reliable solutions in diverse scenarios.

4.6 Life Cycle Management

Software life cycle management encompasses various stages of a software product's life, including inception, design, development, testing, deployment, and maintenance. This process ensures the software meets users' and stakeholders' requirements while adhering to quality standards and cost constraints [9,39]. Differences in life cycle management between machine learning (ML) development and traditional software development arise from ML's reliance on data and iterative model training, as opposed to the more deterministic approach in traditional software development.

In ML development, data is crucial throughout the entire life cycle, with data collection, preprocessing, and feature engineering significantly affecting model performance [64]. Model training in ML development involves iterative experimentation with algorithms, hyperparameters, and data representations [10,31,56]. Validation and testing in ML development involve assessing model generalization to unseen data, which can be challenging due to overfitting and biases [10,31,59]. Deploying an ML model requires serving it in a production environment, monitoring its performance, and updating or fine-tuning it as necessary [8].

Traditional software development has adopted CI/CD practices, but ML development still faces challenges in integrating these practices due to the iterative nature of model training and dependency on data [54]. Emerging tools such as MLflow, TFX, and Kubeflow address the unique requirements of ML CI/CD but still leave room for improvement in aligning these practices with traditional software development [72,8,11]. Ensuring explainable predictions is crucial for gaining user trust and ensuring ethical use in ML development [4,58,43].

Our vision incorporates "What-if" scenarios [57] into ML development to facilitate validation, testing, and interpretability. Sensitivity analysis can help identify potential weaknesses or areas of improvement and inform the selection of features and parameters [63]. Counterfactual explanations provide insights into how a model might behave if specific features or inputs were different, supporting better decision-making and model understanding [70]. Automatic scenario generation techniques allow developers to consider multiple plausible future scenarios and their potential impacts on ML models or software systems to inform model development and decision-making [6,29,46,20,19].

In summary, incorporating what-if scenarios in ML and traditional software development can help bridge the gap between these domains by enhancing model understanding, improving decision-making, and fostering better communication among stakeholders.

4.7 Adaptive Runtime Controller

A runtime controller is a software component responsible for managing and orchestrating the execution of applications or services at runtime. It ensures that the desired state of the system is maintained and adapts to any changes or requirements that may arise during the execution. Kubernetes, a widely adopted platform for orchestrating containerized services, has emerged as a best practice in this context [16]. It offers numerous built-in features and solutions that simplify application deployment, scaling, and management.

Machine learning development and our adaptive module selection require frequent changes in the modules, depending on various factors such as the application's needs, user preferences, or environmental conditions. Kubernetes provides a mechanism called the operator pattern, which can be used to implement this adaptive behavior [23]. An operator is a custom controller that extends the functionality of the Kubernetes API by defining custom resource definitions (CRDs) and implementing custom control logic.

To facilitate the implementation of the adaptive module selection mechanism, we designed our CRD definition to be simple and user-friendly, allowing users to define their requirements with minimal technical knowledge. The controller then translates this high-level desired state into more detailed and advanced technical specifications. Kubernetes CRDs are translated into OpenAPI APIs, enabling seamless integration with the Kubernetes API server.

The controller continually monitors the actual state of the system in the cluster and attempts to match it with the desired state defined by the user. We implemented modules, consisting of both module definition and module implementation, as Kubernetes CRDs. To support deploying modules as a service and adaptive module selection mechanism, we developed several custom controllers that extend the Kubernetes API. These custom controllers manage various aspects of the system, such as container deployment, storage, and communication.

However, since it is impossible to cover every possible deployment need and to ensure the generality of our platform, we designed our CRD in a way that allows more advanced users to develop their own custom controllers. These custom controllers can be used in a pluggable fashion, enabling users to tailor the adaptive runtime controller to their specific needs and requirements. This flexibility allows for a wide range of use cases and applications, making the adaptive runtime controller a powerful tool for managing complex, dynamic systems.

In summary, the adaptive runtime controller leverages the power of Kubernetes and the operator pattern to provide a flexible and extensible platform for managing and orchestrating adaptive module selection in various applications. By designing user-friendly CRDs and supporting custom controllers, the adaptive runtime controller enables users to implement complex adaptive behavior with ease, ultimately leading to more robust and responsive systems.

4.8 Proposed Architecture

To integrate the concepts discussed in the previous subsections, we propose a layered architecture as illustrated in Figure 4. This architecture enables users to

interact with any component in the system, facilitating a seamless user experience. In this section, we will explore the different components of the proposed architecture and their interactions.

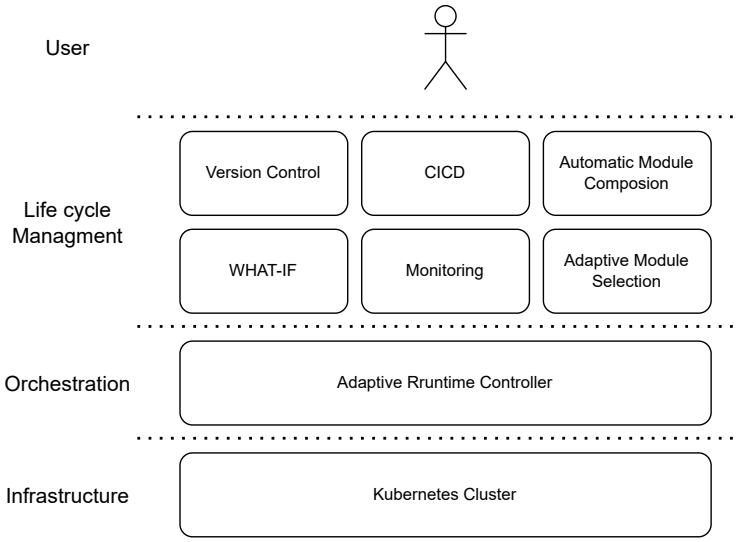

Fig. 4. Proposed Architecture for implementing our vision

Users can add module definitions and module implementations to the system through version control. The ultimate goal is to enable users to request a module definition, add it to the version control system, and trigger the Continuous Integration and Continuous Deployment (CI/CD) pipeline. This pipeline initiates the automatic module composition process, which selects or generates a module implementation for the module definition.

The automatic module composition process creates placeholders for module implementations and leverages adaptive module selection to fill them at design time. This process may involve multiple iterations if a valid selection is not found. If no solution is found, the automatic module composition returns an incomplete module to the user, identifying the missing implementations.

Once a module is ready, meaning that at least one module implementation satisfies the module definition, it is submitted to the adaptive runtime controller for deployment in the cluster. Users can also define monitoring modules and bind them to other modules, such as processing modules. One of the key features of our platform and the "Everything as a Module" (XaaM) vision is the ability to have monitoring modules that monitor other monitoring modules within the cluster.

The what-if component is responsible for generating scenarios for various purposes, such as testing and interpretability. It submits these scenarios to the

adaptive runtime controller, which runs them in separate environments to avoid interference with production systems.

Monitoring modules collect data on the performance of other modules and store it in a standardized format for multiple purposes, such as visualization in a graphical interface or use by the adaptive module selection process. This comprehensive monitoring system ensures that the platform remains efficient, adaptive, and responsive to changing requirements and conditions.

In conclusion, the proposed layered architecture for adaptive module selection seamlessly integrates various components, enabling users to efficiently manage and orchestrate complex systems. By leveraging adaptive runtime controllers, monitoring modules, and what-if components, this architecture provides a robust and flexible foundation for the implementation of the XaaM vision.

5 Conclusion

In this paper, we presented the "Everything as a Module" (XaaM) vision, a comprehensive approach that aims to empower machine learning development by addressing the unique challenges in machine learning and deviations from the best practices of service-oriented software development. We investigated several aspects, identified the gaps, and proposed solutions for bridging these gaps.

We also introduced an architecture to demonstrate how the various components of the XaaM vision can be seamlessly integrated, enabling users to efficiently manage and orchestrate complex systems. We believe that the XaaM vision has the potential to revolutionize the way machine learning systems and software development projects are designed, developed, and maintained, paving the way for more adaptable, efficient, and scalable solutions. By continuing to develop and refine the XaaM vision, we hope to contribute to the effective development of production-grade machine learning applications.

Acknowledgements This research has been sponsored by NWO C2D and TKI HTSM Ecida Project Grant No. 628011003

References

1. Alla, S., Adari, S.K.: What Is MLOps?, pp. 79–124. Apress, Berkeley, CA (2021). https://doi.org/10.1007/978-1-4842-6549-9_3
2. Anand, M.: Cloud monitor: Monitoring applications in cloud. In: 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM). pp. 1–4. IEEE (2012)
3. Arpteg, A., Brinne, B., Crnkovic-Friis, L., Bosch, J.: Software engineering challenges of deep learning. In: 2018 44th euromicro conference on software engineering and advanced applications (SEAA). pp. 50–59. IEEE (2018)
4. Arrieta, A.B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al.: Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. Information fusion **58**, 82–115 (2020)

5. Artrith, N., Butler, K.T., Coudert, F.X., Han, S., Isayev, O., Jain, A., Walsh, A.: Best practices in machine learning for chemistry. Nature chemistry **13**(6), 505–508 (2021)
6. Bankes, S.C.: Tools and techniques for developing policies for complex and uncertain systems. Proceedings of the National Academy of Sciences **99**(suppl_3), 7263–7266 (2002)
7. Barry, D.K., Dick, D.: Chapter 3 - web services and service-oriented architectures. In: Barry, D.K., Dick, D. (eds.) Web Services, Service-Oriented Architectures, and Cloud Computing (Second Edition), pp. 15–33. The Savvy Manager's Guides, Morgan Kaufmann, Boston, second edition edn. (2013). https://doi.org/https://doi.org/10.1016/B978-0-12-398357-2.00003-8, https://www.sciencedirect.com/science/article/pii/B9780123983572000038
8. Baylor, D., Breck, E., Cheng, H.T., Fiedel, N., Foo, C.Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., et al.: Tfx: A tensorflow-based production-scale machine learning platform. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1387–1395 (2017)
9. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al.: Manifesto for agile software development (2001)
10. Bishop, C.M.: Pattern recognition and machine learning, vol. 4. Springer (2006)
11. Bisong, E., Bisong, E.: Kubeflow and kubeflow pipelines. Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners pp. 671–685 (2019)
12. Bodor, A., Hnida, M., Najima, D.: Mlops: Overview of current state and future directions. In: Innovations in Smart Cities Applications Volume 6: The Proceedings of the 7th International Conference on Smart City Applications. pp. 156–165. Springer (2023)
13. Breck, E., Cai, S., Nielsen, E., Salib, M., Sculley, D.: The ml test score: A rubric for ml production readiness and technical debt reduction. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 1123–1132 (2017). https://doi.org/10.1109/BigData.2017.8258038
14. Briese, C., Schlüter, M., Lehr, J., Maurer, K., Krüger, J.: Towards deep learning in industrial applications taking advantage of service-oriented architectures. Procedia Manufacturing **43**, 503–510 (2020)
15. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. Journal of Artificial Intelligence Research **70**, 245–317 (2021)
16. Burns, B., Beda, J., Hightower, K., Evenson, L.: Kubernetes: up and running. " O'Reilly Media, Inc." (2022)
17. Cao, L.: Beyond automl: Mindful and actionable ai and autoai with mind and action. IEEE Intelligent Systems **37**(5), 6–18 (2022)
18. Chaudhary, A., Choudhary, C., Gupta, M.K., Lal, C., Badal, T.: Microservices in Big Data Analytics: Second International, ICETCE 2019, Rajasthan, India, February 1st-2nd 2019, Revised Selected Papers. Springer Nature (2019)
19. Choi, H., Park, S.: A survey of machine learning-based system performance optimization techniques. Applied Sciences **11**(7), 3235 (2021)
20. Deb, K.: Multi-objective optimisation using evolutionary algorithms: an introduction. Springer (2011)
21. Ding, L., Liao, S.: An approximate approach to automatic kernel selection. IEEE Transactions on Cybernetics **47**(3), 554–565 (2016)
22. Ding, Z., Wang, S., Pan, M.: Qos-constrained service selection for networked microservices. IEEE access **8**, 39285–39299 (2020)

23. Dobies, J., Wood, J.: Kubernetes operators: Automating the container orchestration platform. O'Reilly Media (2020)
24. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering pp. 195–216 (2017)
25. Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N.C., Hu, B.: Everything as a service (xaas) on the cloud: Origins, current and future trends. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 621–628 (2015). https://doi.org/10.1109/CLOUD.2015.88
26. Elshawi, R., Maher, M., Sakr, S.: Automated machine learning: State-of-the-art and open challenges. arXiv preprint arXiv:1906.02287 (2019)
27. Fantinato, M., Peres, S.M., Kafeza, E., Chiu, D.K., Hung, P.C.: A review on the integration of deep learning and service-oriented architecture. Journal of Database Management (JDM) **32**(3), 95–119 (2021)
28. Garriga, M., Renzis, A.D., Lizarralde, I., Flores, A., Mateos, C., Cechich, A., Zunino, A.: A structural-semantic web service selection approach to improve retrievability of web services. Information Systems Frontiers **20**, 1319–1344 (2018)
29. Gilbert, N.: Agent-based models. Sage Publications (2019)
30. Gluzmann, P., Panigo, D.: Global search regression: A new automatic model-selection technique for cross-section, time-series, and panel-data regressions. The Stata Journal **15**(2), 325–349 (2015)
31. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
32. Granlund, T., Kopponen, A., Stirbu, V., Myllyaho, L., Mikkonen, T.: Mlops challenges in multi-organization setup: Experiences from two real-world cases. In: 2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN). pp. 82–88 (2021). https://doi.org/10.1109/WAIN52551.2021.00019
33. He, X., Zhao, K., Chu, X.: Automl: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021)
34. Idowu, S., Strüber, D., Berger, T.: Asset management in machine learning: State-of-research and state-of-practice. ACM Computing Surveys **55**(7), 1–35 (2022)
35. Isdahl, R., Gundersen, O.E.: Out-of-the-box reproducibility: A survey of machine learning platforms. In: 2019 15th international conference on eScience (eScience). pp. 86–95. IEEE (2019)
36. Iterative: DVC: Data version control – Git for data & models (2020). https://doi.org/10.5281/zenodo.012345
37. Jordan, M.I., Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects. Science **349**(6245), 255–260 (2015)
38. Kavikondala, A., Muppalla, V., Krishna Prakasha, K., Acharya, V.: Automated retraining of machine learning models. International Journal of Innovative Technology and Exploring Engineering **8**(12), 445–452 (2019)
39. Kim, G., Humble, J., Debois, P., Willis, J., Forsgren, N.: The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations. IT Revolution (2021)
40. Klaise, J., Van Looveren, A., Cox, C., Vacanti, G., Coca, A.: Monitoring and explainability of models in production. arXiv preprint arXiv:2007.06299 (2020)
41. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR) **48**(3), 1–41 (2015)
42. Li, D., Ye, D., Gao, N., Wang, S.: Service selection with qos correlations in distributed service-based systems. IEEE Access **7**, 88718–88732 (2019)
43. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. Advances in neural information processing systems **30** (2017)

44. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning. ACM Computing Surveys (CSUR) **54**(6), 1–35 (2021)
45. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: Machine learning in apache spark. The Journal of Machine Learning Research **17**(1), 1235–1241 (2016)
46. Metropolis, N., Ulam, S.: The monte carlo method. Journal of the American statistical association **44**(247), 335–341 (1949)
47. Moghaddam, M., Davis, J.G.: Simultaneous service selection for multiple composite service requests: A combinatorial auction approach. Decision Support Systems **120**, 81–94 (2019)
48. Mäkinen, S., Skogström, H., Laaksonen, E., Mikkonen, T.: Who needs mlops: What data scientists seek to accomplish and how can mlops help? In: 2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN). pp. 109–112 (2021). https://doi.org/10.1109/WAIN52551.2021.00024
49. Newman, S.: Building microservices. " O'Reilly Media, Inc." (2021)
50. Pachyderm: Pachyderm - scalable, reproducible data science (nd), https://www.pachyderm.com/
51. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003. pp. 3–12. IEEE (2003)
52. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf
53. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research **12**, 2825–2830 (2011)
54. Polyzotis, N., Roy, S., Whang, S.E., Zinkevich, M.: Data management challenges in production machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 1723–1726 (2017)
55. Rabbani, I.M., Aslam, M., Enriquez, A.M.M., Qudeer, Z.: Service association factor (saf) for cloud service selection and recommendation. Information Technology and Control **49**(1), 113–126 (2020)
56. Raschka, S., Mirjalili, V.: Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd (2019)
57. Ravetz, J.R.: The science of 'what-if?'. Futures **29**(6), 533–539 (1997)
58. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1135–1144. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939778, https://doi.org/10.1145/2939672.2939778
59. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Testing machine learning based systems: a systematic mapping. Empirical Software Engineering **25**, 5193–5254 (2020)

60. de la Rúa Martínez, J.: Scalable architecture for automating machine learning model monitoring (2020)
61. Ruf, P., Madan, M., Reich, C., Ould-Abdeslam, D.: Demystifying mlops and presenting a recipe for the selection of open-source tools. Applied Sciences **11**(19), 8861 (2021)
62. Rufino, J., Alam, M., Ferreira, J.: Monitoring v2x applications using devops and docker. In: 2017 International Smart Cities Conference (ISC2). pp. 1–5. IEEE (2017)
63. Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S.: Global sensitivity analysis: the primer. John Wiley & Sons (2008)
64. 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. Advances in neural information processing systems **28** (2015)
65. Silva, S.H., Najafirad, P.: Opportunities and challenges in deep learning adversarial robustness: A survey. arXiv preprint arXiv:2007.00753 (2020)
66. Symeonidis, G., Nerantzis, E., Kazakis, A., Papakostas, G.A.: Mlops - definitions, tools and challenges. In: 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC). pp. 0453–0460 (2022). https://doi.org/10.1109/CCWC54503.2022.9720902
67. Tamburri, D.A.: Sustainable mlops: Trends and challenges. In: 2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC). pp. 17–23. IEEE (2020)
68. Testi, M., Ballabio, M., Frontoni, E., Iannello, G., Moccia, S., Soda, P., Vessio, G.: Mlops: A taxonomy and a methodology. IEEE Access **10**, 63606–63618 (2022). https://doi.org/10.1109/ACCESS.2022.3181730
69. Vartak, M., Madden, S.: Modeldb: Opportunities and challenges in managing machine learning models. IEEE Data Eng. Bull. **41**(4), 16–25 (2018)
70. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: Automated decisions and the gdpr. Harv. JL & Tech. **31**, 841 (2017)
71. Xia, Y., Chen, P., Bao, L., Wang, M., Yang, J.: A qos-aware web service selection algorithm based on clustering. In: 2011 IEEE International Conference on Web Services. pp. 428–435. IEEE (2011)
72. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S.A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., et al.: Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull. **41**(4), 39–45 (2018)
73. Zhou, Y., Yu, Y., Ding, B.: Towards mlops: A case study of ml pipeline platform. In: 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE). pp. 494–500 (2020). https://doi.org/10.1109/ICAICE51518.2020.00102
74. Zolkifli, N.N., Ngah, A., Deraman, A.: Version control system: A review. Procedia Computer Science **135**, 408–415 (2018). https://doi.org/https://doi.org/10.1016/j.procs.2018.08.191, https://www.sciencedirect.com/science/article/pii/S1877050918314819, the 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life