Empowering Machine Learning Development with Service-Oriented Computing Principles

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

[1] Faculty of Science and Engineering, Bernoulli Institute, University of Groningen, Groningen, The Netherlands
{m.hadadian,a.lazovik}@rug.nl

[2] Faculty of Science, Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
v.o.degeler@uva.nl

Abstract. Despite software industries' successful utilization of Service-Oriented Computing (SOC) to streamline software development, machine learning (ML) development has yet to fully integrate these practices. This disparity can be attributed to multiple factors, such as the unique challenges inherent to ML development and the absence of a unified framework for incorporating services into this process. In this paper, we shed light on the disparities between services-oriented computing and machine learning development. We propose "Everything as a Module" (XaaM), a framework designed to encapsulate every ML artifacts including models, code, data, and configurations as individual modules, to bridge this gap. We propose a set of additional steps that need to be taken to empower machine learning development using services-oriented computing via an architecture that facilitates efficient management and orchestration of complex ML 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.

M. Aiello et al. (Eds.): SummerSOC 2023, CCIS 1847, pp. 24–44, 2023.
https://doi.org/10.1007/978-3-031-45728-9_2

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 [25].

In recent years, the software industry has successfully embraced service-oriented computing (SOC) and DevOps ("Development Operations" to automate software development process), which has significantly improved software development processes, enabling better modularity, flexibility, and maintainability [31]. However, ML development has not yet fully adopted these best practices, resulting in a gap between service-oriented computing and ML development [1].

DevOps is a cross-departmental and collaborative endeavor within an organization, aiming to simplify the continuous delivery of new software releases while upholding their integrity and trustworthiness [26]. In service-oriented computing, DevOps is a cultural shift and set of practices for enhancing collaboration between development and operations teams. This approach accelerates the development life cycle for efficient, continuous software service delivery. DevOps incorporates Agile principles, e.g., continuous integration, deployment, and automated testing, which are critical in service-oriented computing for integration and constant service availability. Moreover, DevOps encourages improved communication and collaboration across service production and maintenance teams.

MLOps (Machine Learning Operations) can be considered an extension of DevOps. It applies the principles and practices of DevOps to the specific challenges and requirements of machine learning development. This includes the development, deployment, and lifecycle management of ML models, while addressing the complexities inherent in data-driven machine learning.

Challenges in MLOps stem from the unique nature of machine learning models. For instance, they may degrade over time as data drifts occur, requiring constant monitoring and frequent retraining. Another challenge is managing the lifecycle of a machine learning model, which includes stages like data collection, model training, validation, deployment, and continuous monitoring. Moreover, another common challenge is the reproducibility of ML models due to variations in data, code, configuration, or environment, which can lead to inconsistencies in model performance. MLOps aims to address these challenges by providing a robust framework for managing the ML lifecycle, similar to how DevOps manages the software development lifecycle. It incorporates practices like versioning of datasets and ML models, automated testing, and continuous integration/continuous delivery (CI/CD) for machine learning models to ensure their reliability and performance over time.

In this paper, we investigate the best practices in service-oriented computing and DevOps and identify the gaps between these practices and machine learning development and MLOps. Inspired by Anything as a Service (XaaS), we propose our "Everything as a Module" (XaaM) vision, an enabler of MLOps practices and a comprehensive solution for bridging these gaps. In this vision, we encapsulate every machine learning artifact such as model, code, data, and configurations as a module. XaaM is 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". However, XaaM is different from XaaS which is a business and delivery model that provides various types of services over the Internet. These services could range from Infrastructure (IaaS), Platform (PaaS), to Software (SaaS), and beyond.

The contributions of this research, reflecting our vision, are as follows. First, we present the concept of two levels of modularity, elaborating on the definition of modules, while addressing polymorphism. We then describe our approach to composing complex modules from atomic ones. We also highlight the importance of module versioning for experiment tracking in machine learning applications, with the aim of achieving maximum observability. Here, our intention is to create a system where every artifact is trackable throughout the development process, a goal we strive to achieve through the introduction of module lineage.

Subsequently, we venture into the challenge of monitoring machine learning applications, providing our perspective on managing the lifecycle of modules. This is realized by proposing Adaptive Module Selection and What-if Scenarios. The former aspires to select the most effective module at any given time, while the latter is akin to automatic testing to facilitate improved module evaluation.

This work encapsulates our vision to enhance machine learning development by leveraging the best practices from service-oriented computing, thereby leading to the creation of more robust, efficient, and scalable ML systems. While we anticipate benefits such as improved development efficiency, increased scalability and adaptability of ML applications, and more effective creation of production-grade ML systems, these outcomes depend on a successful implementation of our proposals. One of the motivational goals of our vision is fostering interdepartmental communication and collaboration by advocating a modular design. Note that our work presents a blueprint towards a desired state, not a fully functioning system. We aim to advance machine learning development practices and encourage more widespread use of service-oriented computing in this field.

The rest of this paper is structured as follows: Sect. 2 offers background for a better understanding of service-oriented computing and machine learning development. Section 3 presents a review of related work. Section 4 identifies the gaps that exist in various aspects and proposes solutions to bridge them. Finally, Sect. 5 presents our conclusions.

2 Background

2.1 Service-Oriented Computing

Service-Oriented Computing (SOC) is a distributed computing paradigm that utilizes services as building blocks for applications [31]. These services are often realized as web services or microservices which can be described, published, located, and invoked over a network. This paradigm promotes interoperability, integration, and simplifies large-scale system development, reducing complexity via service reuse and advancing business agility and innovation.

In technical terms, these services are commonly encapsulated within containers, such as Docker, to ensure their isolation and to simplify their deployment and

scaling. Management and orchestration of these containers can be achieved using technologies like Kubernetes, which enables automated deployment, scaling, and management, effectively distributing and coordinating containers across a cluster of machines [8]. Communication among these services, vital for orchestrating complex business processes, leverages protocols like HTTP/REST or gRPC for synchronous, and message brokers for asynchronous communication.

2.2 Machine Learning Development

Drawing from literature [37,41], we synthesize the common elements into a generalized workflow of lifecycle of Artificial Intelligence/Machine Learning (AI/ML) applications depicted in Fig. 1.

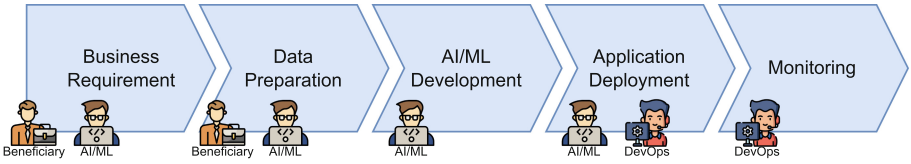

Fig. 1. Coarse-grained AI/ML application life-cycle illustrating the stages and their corresponding actors.

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 integrates models into an application, ensuring its stability, performance, and security in an operational environment. Finally, the *monitoring stage* involves tracking the application's performance, identifying issues, and gathering insights for continuous improvement.

AI/ML development diverges from traditional software development, primarily due to its data-driven nature. It involves iteratively constructing probabilistic models that learn patterns from data, a process that requires extensive experimentation and monitoring. Teams often explore various model architectures and algorithms before settling on a solution. Moreover, decisions are data-driven, emphasizing the quality of the data used for model training. Evaluating a learning model is a complex task, as its performance is tightly coupled with the data. Thus, teams must conduct extensive training and testing on both small and large datasets that closely resemble production data, necessitating a scalable underlying platform. Throughout this process, monitoring and experiment tracking become critical to compare different models, observe their performance over time, and ensure reproducibility. The latter means that running the same model

with the same data should ideally yield the same results, despite the inherent probabilistic nature of ML models. This iterative, monitored, and reproducible process ensures that the AI/ML solution generalizes well to new, unseen data and effectively addresses the defined business requirements.

3 Related Work

This section focuses on MLOps and the study of solutions founded on the principles of service-oriented computing for machine learning. For convenience and cohesion, we discuss other relevant subjects in their sections. Our dual goal is to highlight the differences between ML development and SOC, and to centralize related content for easy access and reference.

MLOps is an emerging paradigm, merging machine learning and traditional software development using DevOps principles. It focuses on automating machine learning development, deployment, and monitoring to boost efficiency and shorten time to market [37]. Testi et al. [41] respond to the fragmented state of MLOps literature by proposing a cohesive taxonomy and standardized methodology for MLOps projects. Several studies address MLOps challenges and solution to facilitate this integration.

Symeonidis et al. [40] delve into the complexities of Machine Learning Operations (MLOps), highlighting challenges like efficient pipeline creation, continual model re-training, comprehensive monitoring, and data manipulation. They discuss tools for data preprocessing, modeling, and operationalization, highlighting AutoML's potential to automate and simplify the machine learning process. Granlund et al. [18] discuss AI/ML operations' integration challenges, focusing on complexities of data consolidation, shared ML model development, and cross-organizational system performance monitoring. They also discuss the scaling challenge related to managing data from multiple entities, developing personalized models, and providing tailored monitoring options in a large, multi-organizational setting. Zhou et al. [44] use existing CI/CD tools and Kubeflow to illuminate potential performance bottlenecks like GPU utilization. Their analysis of time and resource consumption in ML pipelines offers a practical guide for efficient ML pipeline platform construction.

Another interesting area is applying microservices architecture in ML development [10]. Microservices enable the modularization of ML components, allowing for more flexible development and deployment of ML applications. This method also encourages reusability of components, significantly saving development time and effort. Several studies cover machine-learning use for service-oriented computing, but few investigated the integration of machine learning models into service-oriented architectures. Fantinato et al. [15] review the mutual enhancement of service-oriented architecture (SOA) and deep learning. They detail how deep learning aids SOA solutions using web service data and how SOA enables flexible, reusable infrastructures for deep learning. Their study highlights the potential of this synergy for various environments and users, shedding light on these technologies' evolution. Briese et al. [7] propose a service-oriented architecture for rapid deployment of deep learning in reverse logistics, addressing the

problem of uninterpretable markers. Their method allows using ever-expanding, initially small datasets, reducing digitization and labeling costs and time.

Mboweni et al. [28] extensively studied MLOps literature to identify the state-of-the-art and gaps in understanding. Despite abundant literature, their review uncovers a lack of standardization and a shared vision on implementing MLOps across industries, showing a need for further research in this area. While these works offer insights into applying service-oriented computing in machine learning development, they often concentrate on specifics like MLOps or microservices, rather than a holistic vision for enhancing machine learning development using service-oriented computing principles. In this paper, we aim to broaden the perspective on this topic, discussing a variety of techniques 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 our perspective of modules and explaining module composition since these components form the fundamental pillars of our vision. Prior to delving into other constituent components, we present an overarching overview of our system. Subsequently, we explain each component in a more intricate manner.

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 assembled, 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[1] or PyTorch[2] 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 modularity offers greater flexibility, improved

[1] https://scikit-learn.org/.
[2] https://pytorch.org/.

maintainability, and enhanced scalability, ensuring that the ML application remains adaptable to changing requirements and emerging technologies.

Due to the wealth of available frameworks in machine learning 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 [29]. 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 ML 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 Fig. 3a. It is important to note that the definitions and implementations of the modules are loosely coupled. If a mod-

ule 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. 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. The second implementation, "MinMaxScaler" Rescales the input features to a specified range (usually [0, 1]). Although both implementations are scaling the input features, their output has a different distribution.

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 an ML system, including executable codes, ML pipelines, and datasets. Figure 2 offers a demonstrative example of XaaM for the training and inference stages of an ML application, 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.

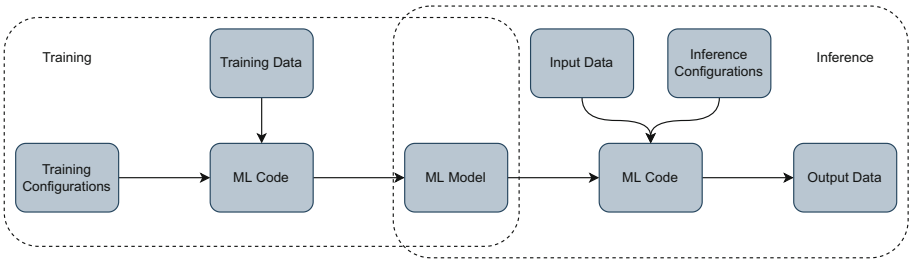

Fig. 2. An example XaaM demonstration for training and inference

4.2 Module Composition

Module composition is the process of combining simpler modules 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 [3], 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 Fig. 3c.

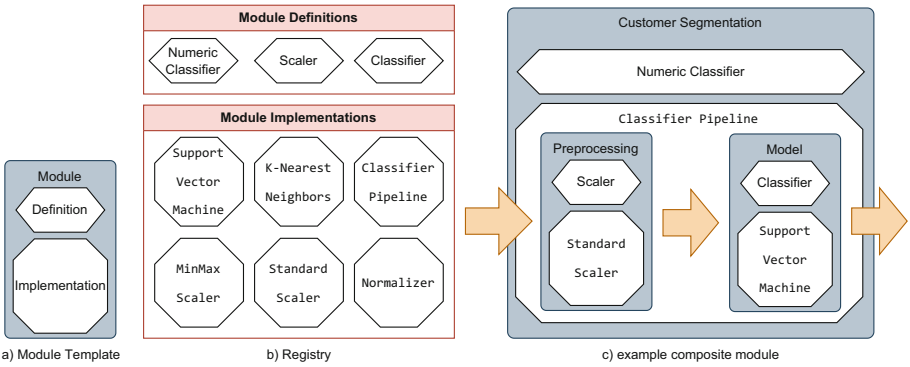

Fig. 3. Module Composition Example: a) Module template, b) Sample registry, c) A composite module from registry elements, adhering to the given template.

Our framework houses module definitions and implementations in a registry (Fig. 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 [43] or Directed Acyclic Graph (DAG) [5] 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, 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 [9] and AutoML [19]. While these approaches hold significant promise, they are not without their challenges, as identified by Elshawi et al. [14]. 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 Proposed Architecture

We propose a simplified layered architecture that serves as an overview of the implementation of our vision illustrated in Fig. 4. The ultimate goal is to enable users to define module definitions and the system takes care of the rest including automatically associating it with proper implementations and deploying the modules. In this section, we will explore the different components of the proposed architecture and their interactions.

The version control component is the entry point to the system where users can add module definitions and module implementations to the registry. A proper versioning mechanism enables observability, extensive experiment tracking, and module lineage. In particular, it is essential for other components such as monitoring and adaptive module selection to fully operate. Therefore, to avoid overcomplicating other components' mechanisms, we propose a unified versioning mechanism for every module.

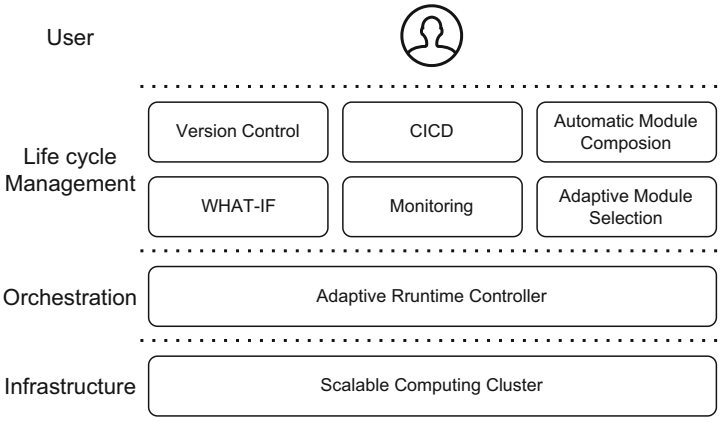

Fig. 4. A layered architecture implements the XaaM vision, with items in each layer interacting only with adjacent layer items.

From this point the Continuous Integration and Continuous Deployment (CI/CD) pipeline is responsible for managing the automatic workflow of delivering module definitions from the registry to an environment among other responsibilities. This pipeline invokes the automatic module composition and selection components in order to complete modules by selecting or generating module implementations for the module definitions while testing different scenarios via the What-If component. Finally, it invokes the adaptive runtime controller for deploying the module on the underlying infrastructure.

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 inference modules. One of the key features of our platform and the 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. The adaptive module selection component actively checks the monitoring data and updates the module implementations accordingly to satisfy the requirement that may come from the user or a downstream module. This

comprehensive monitoring system ensures that the platform remains efficient, adaptive, and responsive to changing requirements and conditions.

The adaptive runtime controller component is responsible for continuously syncing the actual state of the scalable computing cluster with the desired state. We develop our control mechanism on top of Kubernetes which is an open-source container orchestration system that can operate on a diverse range of infrastructures such as Amazon AWS[3] and Google Cloud Platform[4]. This implementation makes our system portable and avoids vendor lock-in. Finally, the scalable computing cluster is where the modules run.

4.4 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 [45], and tracking changes in data, models, hyperparameters, and code in the latter [6]. 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 such as lack of standardization [21] and intricacies involved in tracking and recording model changes, leading to problems like reproducibility and model comparison [20].

To address these challenges, specialized version control systems such as DVC[5], Pachyderm[6], and MLflow[7] 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, i.e., YAML (YAML Ain't Markup Language), enabling Git-based version control on a metadata level. Module definitions are stored fully in YAML format since they are designed to be describable in text format. On the other hand, module implementations are more complex as they can have various forms. Therefore, we designed a unified YAML description that captures features of the implementation. This YAML file is then linked to the actual implementation.

[3] https://aws.amazon.com/.
[4] https://cloud.google.com/.
[5] https://dvc.org/.
[6] https://www.pachyderm.com/.
[7] https://mlflow.org/.

Within our XaaM vision, we adopt a modular approach where every ML asset encompassing data, code, models, and executables are treated as distinct modules, thereby ensuring a cohesive framework across diverse projects and teams. We establish a registry of module definitions and implementations in an informative text format, specifically YAML, which facilitates version control through Git at a metadata level. The module definitions are exclusively stored in YAML to align with their text-based descriptive nature. Conversely, module implementations necessitate a more comprehensive treatment due to their multifaceted nature. To address this, we have devised a comprehensive YAML schema that encapsulates the nuances of the implementation. This specialized YAML file is linked to the tangible implementation, facilitating a unified and coherent framework. This method streamlines project administration, guarantees uniformity, and eases the integration of varied modules and tools, thereby fostering the creation of intricate, scalable machine learning solutions.

Our cohesive XaaM versioning approach significantly enhances both transparency and reproducibility within the context of module development. This is achieved through the comprehensive preservation of module lineage, encompassing every facet from data and code to configuration settings, that contributed to creating each module. Put simply, this framework empowers users to carefully trace the trajectory and evolution of individual modules within the project. For example, users possess the capability to discern the origins of output data, encompassing details such as the model employed, configuration parameters utilized, and input data employed during its generation.

Ultimately, our approach allows for the targeted application of techniques aligned with each unique implementation, all within the framework of our YAML-based versioning system. 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.

4.5 Continuous Monitoring

Continuous monitoring is vital for managing application health and performance in software development projects, particularly in service-based applications with complex interdependencies [13]. In machine learning projects, continuous monitoring is even more critical since it ensures model performance remains consistent, detects data drift, anomalies, and performance issues [41], and determines when model retraining is necessary [22].

Challenges in continuous monitoring include integrating monitoring metrics and KPI evaluations from different teams [41], scalability and real-time monitoring [39], addressing the statistical nature of drift detection and outlier identification [24], standardization data collection and storage methods [30], and monitoring upstream processes that feed data to ML systems [39]. 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. Monitoring modules are treated almost the same way as other modules. The only difference is the way of handling modules by the adaptive runtime controller which employs specific mechanisms to collect data from other modules seamlessly and redirect the output data to a standard storage. Subsequently, the monitoring team can develop the monitoring module definition and implementation in the same way as other modules. They can also benefit from the automation offered by our system to create composite monitoring modules automatically that deliver the desired functionality. It is also worth mentioning that since the monitoring modules are the same as other modules, they can also be monitored using other monitoring modules.

We introduce the concept of Monitoring as a Module, which serves as an integration point for diverse monitoring methodologies, fostering harmonious collaboration among teams. Monitoring modules are treated almost the same way as other modules. A nuanced distinction lies in the manner by which these modules interface with the adaptive runtime controller. This controller employs distinct mechanisms to seamlessly fetch information from concurrent modules, channeling output data to a standardized repository.

Consequently, the monitoring team finds themselves capacitated to devise module definitions and implementations for monitoring on par with general module practices. Leveraging the automation inherent to our framework, they are further empowered to fabricate composite monitoring modules. Moreover, it is worth mentioning that the equivalence of monitoring modules with other modules extends to the realm of monitoring, wherein monitoring modules themselves are amenable to oversight through analogous monitoring modules. This confers the capability for adaptive module selection to not merely ensure the fulfillment of requirements by the monitored modules, but also to maintain the monitoring modules' correctness. Ultimately, this synergy culminates in an enhanced performance exhibited by the monitored modules.

4.6 Adaptive Module 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. This process can occur at three stages in the module composition life cycle: design time, deployment time, and runtime.

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.

Service selection algorithms prioritize QoS attributes such as response time, success rate, and cost [11]. In machine learning development, we must ensure QoS metrics while satisfying performance requirements, like accuracy and Mean Squared Error (MSE) [17]. Addressing the interdependence of metrics is crucial, considering modules correlations and user requirements correlations [27,33].

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

- **candidating:** find suitable module implementations for a module definition.
- **ranking**: order top-performing module implementations for a scenario.
- **choosing**: determine if updating the production module is worthwhile.

Challenges during the *candidating* 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. In our vision, find-matching algorithms' input is the YAML-based descriptions of module definition and implementation. Analogous to web-service selection [16], we incorporate the structural-semantic approach enhances the candidate identification phase, using domain ontology concepts, similarity measures, and structural properties analysis to select suitable module implementations.

Defining the scenario presents a significant challenge during the *ranking* stage, given its reliance on variable factors such as incoming data streams, sensing and operational environments, and requirements. The subsequent difficulty lies in forecasting future scenarios and the corresponding performance of each individual module within these hypothetical situations. In our vision, we propose a pipeline incorporating a scenario prediction algorithm and a metamodel. This metamodel is designed to estimate module performance utilizing the historical data collected via monitoring modules.

Finally, the *choosing* phase entails the critical decision of whether to update the existing module in the production environment, taking into account the costs of redeployment and possible non-optimal performance measures. The effectiveness of this phase is inextricably linked to the successful execution of the ranking stage and the accuracy of future scenario prediction. Furthermore, it must take into account the distinct overheads and deployment costs that may be associated with different modules.

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 scenarios and maintain optimal performance across various scenarios.

4.7 Life Cycle Management

Software life cycle management covers stages from inception to maintenance of a software product. This process ensures software meets user and stakeholder

requirements, adheres to quality standards, and fits cost constraints [23]. Differences in life cycle management between machine learning (ML) and traditional software development stem from ML's data reliance and iterative model training, contrasted with traditional software development's deterministic approach.

In ML development, data is crucial throughout the entire life cycle, with data collection, preprocessing, and feature engineering significantly affecting model performance [39]. Model training in ML development involves iterative experimentation with algorithms, hyperparameters, and data representations [34]. Validation and testing in ML development involve assessing model generalization to unseen data, which can be challenging due to overfitting and biases [36]. Deploying an ML model requires serving it in a production environment, monitoring its performance, and updating or fine-tuning it as necessary [4].

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 [32]. Emerging tools such as MLflow [43], TFX [4], and Kubeflow [5] address the unique requirements of ML CI/CD but still leave room for improvement in aligning these practices with traditional software development. Ensuring explainable predictions is crucial for gaining user trust and ensuring ethical use in ML development [2].

Our vision incorporates "What-if" scenarios [35] 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 [38]. Counterfactual explanations provide insights into how a model might behave if specific features or inputs were different, supporting better decision-making and model understanding [42]. 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.

The What-If scenario component plays a crucial role in our vision, encompassing a range of possibilities that significantly enhance our machine learning module's adaptability and robustness.

Firstly, It engages with the aspect of "new data vs current modules". In essence, it consistently explores the hypothetical question, "What if an alternative equivalent module was operational instead of the currently running one?" This means that it generates various scenarios where any active module is replaced by an equivalent alternative, contributing to the system's adaptability.

Secondly, it contemplates "new modules vs historical data". Whenever a new module is introduced, the component seeks to answer, "What if these modules were already integrated into our system?" This implies that it measures the performance of the newcomer against collected historical data. This is accomplished either by running the new module or estimating its performance. This provision of the What-If scenario component aids in adaptive module selection, generating more data to enhance performance.

Lastly, it examines the interaction of "current modules vs unseen data". Unlike traditional software development, machine learning development does not allow for extensive testing. To counter this limitation, the What-If scenario component learns from the shortcomings of other equivalent modules, generating corner-case scenarios to test modules against any unforeseen circumstances. This not only ensures model robustness but also facilitates proactive identification of potential issues, significantly improving the system's resilience.

This component greatly aid in the process of adaptive module selection, enabling the system to collect more data and consequently perform more effectively. Ultimately, incorporating what-if scenarios in ML and traditional software development can help bridge the gap between these domains by enhancing model understanding and improving decision-making.

4.8 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 [8]. 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 [12]. An operator is a custom controller that extends the functionality of the Kubernetes API by implementing custom control logic and defining custom resource definitions (CRDs), which are stored in YAML format.

We designed CRDs to possess a one-to-one correspondence with module definitions and implementation, thereby making them straightforward and user-friendly. This allows users to specify their requirements with minimal technical acumen. In other words, the user-defined module definitions and implementations serve as the direct input to the controller, which represents the desired state. The controller then translates this high-level desired state into 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 modules 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.

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. 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)
2. Arrieta, A.B., et al.: Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Inf. Fusion **58**, 82–115 (2020)
3. 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 (2013)

4. Baylor, D., 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)
5. Bisong, E., Bisong, E.: Kubeflow and kubeflow pipelines. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, pp. 671–685 (2019)
6. 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). https://doi.org/10.1007/978-3-031-26852-6_14
7. Briese, C., Schlüter, M., Lehr, J., Maurer, K., Krüger, J.: Towards deep learning in industrial applications taking advantage of service-oriented architectures. Procedia Manuf. **43**, 503–510 (2020)
8. Burns, B., Beda, J., Hightower, K., Evenson, L.: Kubernetes: up and running. O'Reilly Media, Inc. (2022)
9. Cao, L.: Beyond AutoML: mindful and actionable AI and AutoAI with mind and action. IEEE Intell. Syst. **37**(5), 6–18 (2022)
10. 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). https://doi.org/10.1007/978-981-15-0128-9
11. Ding, Z., Wang, S., Pan, M.: QoS-constrained service selection for networked microservices. IEEE Access **8**, 39285–39299 (2020)
12. Dobies, J., Wood, J.: Kubernetes operators: automating the container orchestration platform. O'Reilly Media (2020)
13. Dragoni, N., et al.: Microservices: yesterday, today, and tomorrow. Present Ulterior Softw. Eng., 195–216 (2017)
14. Elshawi, R., Maher, M., Sakr, S.: Automated machine learning: state-of-the-art and open challenges. arXiv preprint arXiv:1906.02287 (2019)
15. 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. J. Database Manage. (JDM) **32**(3), 95–119 (2021)
16. Garriga, M., et al.: A structural-semantic web service selection approach to improve retrievability of web services. Inf. Syst. Front. **20**, 1319–1344 (2018)
17. Gluzmann, P., Panigo, D.: Global search regression: a new automatic model-selection technique for cross-section, time-series, and panel-data regressions. Stand Genomic Sci. **15**(2), 325–349 (2015)
18. 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)
19. He, X., Zhao, K., Chu, X.: AutoML: a survey of the state-of-the-art. Knowl.-Based Syst. **212**, 106622 (2021)
20. Idowu, S., Strüber, D., Berger, T.: Asset management in machine learning: state-of-research and state-of-practice. ACM Comput. Surv. **55**(7), 1–35 (2022)
21. 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)
22. Kavikondala, A., Muppalla, V., Krishna Prakasha, K., Acharya, V.: Automated retraining of machine learning models. Int. J. Innov. Technol. Explor. Eng. **8**(12), 445–452 (2019)

23. 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)
24. Klaise, J., Van Looveren, A., Cox, C., Vacanti, G., Coca, A.: Monitoring and explainability of models in production. arXiv preprint arXiv:2007.06299 (2020)
25. Kreuzberger, D., Kühl, N., Hirschl, S.: Machine Learning Operations (MLOps): overview, definition, and architecture. IEEE Access **11**, 31866–31879 (2023)
26. Leite, L., Rocha, C., Kon, F., Milojicic, D., Meirelles, P.: A survey of devops concepts and challenges. ACM Comput. Surv. **52**(6) (2019)
27. Li, D., Ye, D., Gao, N., Wang, S.: Service selection with QoS correlations in distributed service-based systems. IEEE Access **7**, 88718–88732 (2019)
28. Mboweni, T., Masombuka, T., Dongmo, C.: A systematic review of machine learning devops. In: 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), pp. 1–6. IEEE (2022)
29. 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)
30. Newman, S.: Building Microservices. O'Reilly Media, Inc. (2021)
31. 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)
32. 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)
33. Rabbani, I.M., Aslam, M., Enriquez, A.M.M., Qudeer, Z.: Service association factor (SAF) for cloud service selection and recommendation. Inf. Technol. Control **49**(1), 113–126 (2020)
34. Raschka, S., Mirjalili, V.: Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd (2019)
35. Ravetz, J.R.: The science of 'what-if?'. Futures **29**(6), 533–539 (1997)
36. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Testing machine learning based systems: a systematic mapping. Empir. Softw. Eng. **25**, 5193–5254 (2020)
37. Ruf, P., Madan, M., Reich, C., Ould-Abdeslam, D.: Demystifying MLOps and presenting a recipe for the selection of open-source tools. Appl. Sci. **11**(19), 8861 (2021)
38. Saltelli, A., et al.: Global sensitivity analysis: the primer. John Wiley & Sons (2008)
39. Sculley, D., et al.: Hidden technical debt in machine learning systems. In: Advances in Neural Information Processing Systems 28 (2015)
40. 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)
41. Testi, M., et al.: MLOps: a taxonomy and a methodology. IEEE Access **10**, 63606–63618 (2022)
42. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: automated decisions and the GDPR. Harv. JL Tech. **31**, 841 (2017)
43. Zaharia, M., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. **41**(4), 39–45 (2018)

44. 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)
45. Zolkifli, N.N., Ngah, A., Deraman, A.: Version control system: a review. In: Procedia Computer Science, the 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018): Empowering Smart Technology in Digital Era for a Better Life, vol. 135, pp. 408–415 (2018)