Article Type: Bridging the Gap Between Machine Learning and Operations

Towards an open-source MLOps architecture

Antonio M. Burgueño-Romero, *KHAOS Research group, ITIS Software, Universidad de Málaga*
Antonio Benítez-Hidalgo, *KHAOS Research group, ITIS Software, Universidad de Málaga*
Cristóbal Barba-González, *KHAOS Research group, ITIS Software, Universidad de Málaga*
José F. Aldana-Montes, *KHAOS Research group, ITIS Software, Universidad de Málaga*

Abstract—The need for robust MLOps frameworks has become paramount in a world increasingly reliant on AI. Current solutions range from proprietary cloud services to independent open-source components, each with advantages and limitations. This paper presents a Kubernetes-based, open-source MLOps framework designed to streamline the lifecycle management of machine learning models in production environments. It integrates a comprehensive suite of open-source tools compatible with Python, covering all aspects from development, testing, deployment, and monitoring to updating models, reducing the need for human intervention. Finally, we compared state-of-the-art MLOps tools and frameworks, demonstrating that our framework meets the same features as proprietary options, such as Amazon SageMaker.

Currently, we live in a world where the domain of artificial intelligence (AI) encompasses nearly every aspect of modern life. Companies and organizations are progressively incorporating predictive algorithms into their services, including facial recognition technologies, sales prediction mechanisms, and intelligent conversational systems, among others.

Nonetheless, leading tech companies have acknowledged the need to maintain, provide access to, and scale these machine learning (ML) models in the same way that has traditionally been done with any microservice [1]. Consequently, by applying traditional DevOps principles to ML models, machine learning operations (MLOps) emerged.

One of the main goals of MLOps is to design a framework or workflow that automatically manages the lifecycle of a set of ML models without human intervention. This includes everything from development, testing, and deployment to monitoring and updating the models in production environments.

Current MLOps frameworks range from paid end-to-end cloud solutions to several independent open-source (OS) services [2]. Most of these frameworks are compatible with Kubernetes, given the containerized architecture inherent to microservices and its advantageous features, such as self-healing and autoscaling.

Using paid cloud services like Azure Machine Learning [3] or Amazon SageMaker [4] offers many advantages, as they are professional services. However, utilizing these services also has several disadvantages, apart from the apparent cost. These frameworks are black boxes, not modular, which can lead to issues when project directions shift or preferences change, leaving users limited flexibility and control over their ML workflows.

In this scenario, an OS alternative may be the option for those seeking complete control over their project's management. The modularity, flexibility, and transparency of emerging OS solutions are unparalleled [5]. Yet, it's important to note that many of these OS services function as independent entities rather than as a cohesive MLOps framework.

An ideal solution promotes a unified OS MLOps architecture to make ML lifecycle management more streamlined, accessible, and sustainable. This approach emphasizes encouraging community contributions and ensuring the system remains flexible enough to adapt to new ML advancements.

Although the overview of an MLOps infrastructure and core capabilities is strongly defined [6], we still

Digital Object Identifier 10.1109/XXX.0000.0000000

cannot see an OS solution that implements all the components and characteristics defined. Numerous works have made progress in the field, providing partial solutions to the problem. CodeReef [7] introduced an open platform addressing the deployment and benchmarking of ML models in production environments. OCDL (One Click Deep Learning) [8] provides an OS suite of tools that facilitates encapsulation, resource sharing, model versioning, and model deployments. More recent works, such as LinkEdge [9], develop specific frameworks for concrete use cases, such as internet of things (IoT) edge devices. The framework automates data collection and model training, as well as the monitoring and deployment of the model. Recent reviews, such as [10], compare OS and enterprise tools for an MLOps framework.

However, proprietary tools exist that meet all the highlighted features. Although CodeReef and OCDL present as open platforms for portable MLOps, they primarily focus on deploying and versioning ML models. LinkEdge expands on this by incorporating additional components, such as monitoring and model registry, into its infrastructure. However, despite being a slightly more comprehensive OS framework, it lacks documentation or examples, which can be a significant barrier for users looking to implement and utilize its features effectively. Previous works also develop framework components in-house rather than leveraging existing OS solutions. This approach can limit the potential for creating more integrated and evolutionary MLOps platforms that benefit from community-driven improvements. Utilizing community-supported components creates a more significant opportunity for platforms to evolve and improve, suggesting a shift towards more collaborative development practices in the MLOps field.

In this article, we introduce a Kubernetes-based MLOps framework that is easy to deploy and entirely composed of OS tools compatible with Python programming. This framework encompasses all the capabilities that, according to [6], an end-to-end MLOps workflow should have, as well as the features considered in reviews [2], [10]. This approach aims to provide a comprehensive, community-supported solution to streamline the whole lifecycle of ML models in production environments. The main contribution of this work is the integration of all OS components into a user-friendly framework, complete with documentation for deployment and providing various examples across different use cases in diverse fields. This presents a foundation and a step forward in the field of MLOps, focusing on OS technologies.

In the rest of the paper, we describe the architecture and implementation of the framework, discuss our experience using it in various use cases, and validate its functionality.

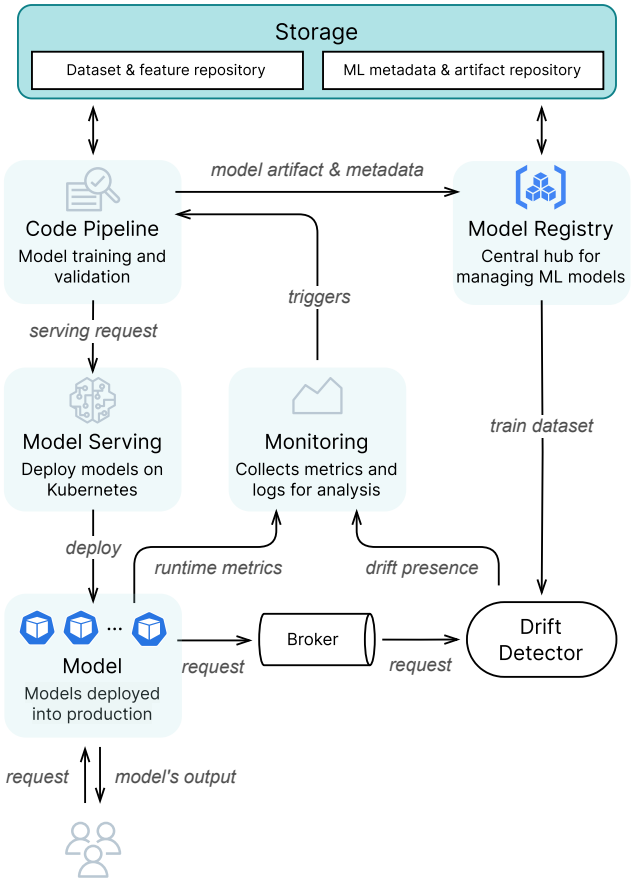

FIGURE 1. Overview of the proposed MLOps architecture.

ARCHITECTURE AND IMPLEMENTATION

The core architecture of the MLOps framework is depicted in Figure 1. The documentation, files, and packages needed for infrastructure deployment are distributed under a free and OS license at https://github.com/KhaosResearch/mlops-infra.

First, the central repository enabling the registration, reproducibility, and versioning of ML models is known as the **model registry**. We have chosen the OS tool **MLflow** [1] as our model registry for its Python API, intuitive graphical user interface (GUI), and extensive features. It has been connected to a cloud-native S3-

[1] https://mlflow.org/

compatible object storage (i.e., MinIO[2]) for storing blobs, and to a relational database (PostgreSQL) for storing metadata.

Second, the **model serving platform** responsible for deploying models on Kubernetes is **Seldon Core** [3]. This tool simplifies converting ML models into containerized production APIs using Kubernetes custom resource definitions (CRDs), automatically generating runtime metrics and facilitating the integration of custom metrics. Seldon offers a simple configuration for horizontal auto-scaling of models, proving particularly beneficial in an environment that handles a substantial volume of prediction requests. Finally, another feature that led us to choose Seldon Core as our model serving platform was the ease of conducting canary tests, A/B testing, and shadow deployments.

The third component of the framework, the **pipeline orchestrator**, holds paramount significance, especially during the execution of the model retraining pipeline. This component simplifies the definition of task workflows and automates their execution in a containerized environment, thereby reducing or eliminating the need for human intervention. **Prefect Core** [4] has been implemented as the pipeline orchestrator in the current environment. We have chosen Prefect because it is made by and for Python code. Prefect allows for the easy organization and automation of Python code. To execute tasks on Kubernetes, workers are defined as lightweight polling services responsible for receiving scheduled work. Prefect connects to the same PostgreSQL database to which the model registry is connected, storing everything necessary to ensure the reproducibility of any execution. In addition, Prefect offers a comprehensive GUI, allowing for workflow execution, real-time visualization, and scheduling, among many other available functionalities.

The fourth component is the **drift detector**. Seldon's Alibi Detect library provides Python implementation of typical drift detection algorithms, both online and offline. However, no dedicated OS tool easily integrates them into a framework. Therefore, we have created a simple deployment that connects these drift algorithms with the deployed models via **Kafka** [5] topics. Once an ML model predicts one or a batch of data, both the prediction request and the data are sent to Kafka (whose brokers scale about user requests). The deployed drift algorithms will use this data to compute and export metrics, which will help determine when to retrain the model.

Lastly, the **monitoring system** completes the workflow cycle. This component is responsible for scraping, visualizing, and analyzing the metrics exported by ML models and drift detectors. Moreover, it plays a crucial role in triggering alerts when the metrics are not as expected, indicating to the developer that something is not functioning correctly or that the model needs to be retrained (or even directly triggering the retraining pipeline). We have used a combination of Prometheus [6] and Grafana [7] as the monitoring and alert manager system since they are acclaimed for their robustness, flexibility, and wide adoption in the industry.

The architecture implementation has been designed to be flexible, modular, scalable, and distributable. Kubernetes was selected as the orchestration system to provide these features, along with seamless deployment facilitated by the **Helm** package manager. This architecture's adaptability extends to deployment across both on-premise environments and cloud platforms. Such versatility ensures that organizations can harness its benefits regardless of their infrastructure preferences, whether prioritizing the security and control of on-premise solutions or the scalability and flexibility of cloud services. Moreover, the OS nature of this system is tailored for customization, enabling users to adjust deployments to suit their specific requirements.

The choice of Kubernetes does not mean the architecture is tied to it. Some users might prefer using Docker Swarm, which offers a simpler setup. Additionally, the services used have been chosen for their flexibility, but any component can be replaced with other open-source services offering similar functionalities, if desired. For example, Seldon Core can be replaced with TensorFlow Serving [8], which is optimized for TensorFlow models, offering improved performance and easier integration but limiting deployment to only these types of models. Similarly, Apache Airflow [9] and Neptune.ai [10] are good open-source alternatives for pipeline orchestration and model registry, respectively.

In any case, once the architecture has been deployed on the desired infrastructure, the next step is integrating an ML model into the framework. This

[2] https://min.io/
[3] https://www.seldon.io/
[4] https://www.prefect.io/
[5] https://kafka.apache.org/
[6] https://prometheus.io/
[7] https://grafana.com/
[8] https://www.tensorflow.org/
[9] https://airflow.apache.org/
[10] https://neptune.ai/

involves several tasks for the developer to perform:

- Upload a base version of the model to the model registry (MLflow).
- Encapsulate the ML model using Seldon's Python wrapper.
- Upload the training pipeline to the pipeline orchestrator (Prefect).
- Configure rules in Grafana to trigger alerts when desired conditions are met.

This process requires minimal effort, considering the significant benefits that this MLOps framework provides to the lifecycle of deployed models. Consequently, end-users can interact with the ML model through the API exposed by the model serving. The model instances will dynamically scale based on user demand. At the same time, the model will provide runtime metrics, while the drift detector will generate additional metrics related to data drift. Suppose the monitoring system identifies that the metrics meet any predefined alarm conditions. In that case, the developer can retrain the model with new data by simply instructing the pipeline orchestrator (this process can also be automated). If the model is retrained, decisions can be made whether to directly replace it, perform shadow deployment to validate it in production, or even conduct A/B testing or canary testing.

USE CASES

Given the versatility of the MLOps framework and its Python-based approach for coding all aspects of the model, it applies to a wide range of use cases. This includes everything from training pipelines for classification models on tabular data to more complex pipelines for classification models dealing with mixed data requiring extensive pre-processing.

The pipeline orchestrator Prefect, offers the flexibility to define training pipelines that deal with complex data, even if they involve processing natural language data, images, or tabular data. Additionally, it allows for specific designation of GPU requirements if any part of the process necessitates GPU utilization. The encapsulated model can also pre-process data before making predictions if required. Currently, the only limitation lies in the drift detector, which is confined to the algorithms implemented by Seldon's Alibi Detect library. While this package supports drift detection for most data types, drift detection for natural language data is currently unavailable. Nonetheless, drift detection remains optional, as runtime metrics are valid indicators for identifying retraining needs.

Examples already implemented with this MLOps framework include tabular data classification (available at https://github.com/KhaosResearch/mlops-creditcard) and multi-year land cover prediction using satellite imagery (available at https://github.com/KhaosResearch/mlops-landcoverpy), to name a few.

As can be seen, the potential of this MLOps framework lies in the programming capabilities of developers, as beyond that, it is boundless.

COMPARISON

Several recent reviews [2], [10] have compared state-of-the-art MLOps tools, including OS and proprietary options. Among these, we have already discussed Amazon SageMaker and MLflow (which is partially used in our framework as a model registry). Kubeflow [11], Iterative Enterprise [12], DataRobot [13], ClearML [14], MLReef [15], and Streamlit [16] are also compared in the review.

Table 1 states a comparison of functionalities of each MLOps tool. These functionalities were chosen based on their critical role in the machine learning lifecycle. Hyperparameter tuning (HT) assesses the ability to optimize model parameters. Data versioning (DV) and pipeline versioning (PV) are essential for reproducibility and tracking changes. Model deployment (MD) ensures models are accessible in production. CI/CD availability (CC) supports continuous integration and deployment, enhancing development efficiency. Performance monitoring (PM) is crucial for tracking model performance and detecting issues. Model and experiment versioning (MV) is necessary for organizing and retrieving past experiments. OS code availability (OS) guarantees transparency and adaptability.

Like AWS SageMaker, our proposed framework meets the criteria for a comprehensive MLOps solution. The asterisk (*) in Table 1 indicates that the feature requires the development of a pipeline in Prefect for performing hyperparameter tuning during the experimentation phase. This can be easily achieved using dedicated Python libraries. The rest of the features are fully met:

Data versioning is implemented by storing the datasets used for each new model version via MLflow.

[11] https://www.kubeflow.org/
[12] https://iterative.ai/
[13] https://www.datarobot.com/
[14] https://clear.ml/
[15] https://about.mlreef.com/
[16] https://streamlit.io/

TABLE 1. Feature comparison of existing platforms along with our proposal.

	HT	DV	PV	MD	CC	PM	MV	OS
Kubeflow	✓		✓		✓	✓	✓	✓
MLflow	✓	✓		✓	✓	✓	✓	✓
It. Enterprise			✓	✓	✓	✓		✓
DataRobot	✓		✓		✓	✓	✓	
ClearML	✓	✓	✓		✓	✓		✓
MLReef		✓	✓		✓	✓	✓	✓
Streamlit		✓	✓		✓	✓	✓	✓
SageMaker	✓	✓	✓	✓	✓	✓	✓	
Our proprosal	✓*	✓	✓	✓	✓	✓	✓	✓

Pipeline versioning is managed in Prefect and updates whenever the pipeline code is modified. Model deployment is facilitated through Seldon Core. CI/CD functionality is achieved using either GitLab or GitHub, allowing for automatic updates to the pipeline version or the model wrapper when the code is updated and conducting tests. Prometheus handles performance monitoring by collecting metrics exported by both Seldon and the drift detector. Model and experiment versioning is also handled via MLflow, storing the model and everything related to the training phase (i.e., parameters and metrics). OS code availability is ensured, as both the infrastructure deployment and the code for each component are OS.

While integrating multiple tools may initially seem daunting and potentially costly, the resulting benefits justify the effort. By carefully selecting the best OS tool for each specific functionality, we enhance the lifecycle of every ML model.

LIMITATIONS

The most notable limitation of the framework lies in the drift detection component, an area of research that has recently gained practical importance with the rise of MLOps. Currently, drift models cannot be serialized and saved as objects, requiring them to be retrained each time an instance is initialized. This is suboptimal because if the model input has a high dimensionality or the dataset is large, initialization can be slow and resource-intensive, necessitating larger container sizes. However, this limitation is expected to be resolved as more efficient drift detection solutions emerge in the open-source community.

CONCLUSION

We presented in this paper an OS, modular MLOps architecture, oriented towards Python code, which reduces human intervention in the lifecycle of ML models. By now, the architecture fulfills the same MLOps functionalities as the professional options offered by large companies. That has been achieved by leveraging the most advanced OS components available today, which benefit from significant participation from their active community. We will study in the future the use of the framework using Large Language Models (LLMs). To conclude, the OS nature of the framework will lead to an increasing number of examples and documentation, thereby facilitating its use for new users.

ACKNOWLEDGMENTS

This work has been funded by grant PID2020-112540RB-C41 (MCIN/AEI/10.13039/501100011033/), AETHER-UMA (A smart data holistic approach for context-aware data analytics: semantics and context exploitation) and the Junta de Andalucía, Spain, under contract QUAL21 010UMA.

REFERENCES

1. I. Karamitsos, S. Albarhami, C. Apostolopoulos, Applying devops practices of continuous automation for machine learning, Information 11 (7) (2020) 363.
2. N. Hewage, D. Meedeniya, Machine learning operations: A survey on mlops tool support, arXiv preprint arXiv:2202.10169 (2022).
3. Azure machine learning: Build and deploy ml models.
   URL https://azure.microsoft.com/en-us/products/machine-learning/
4. Amazon sagemaker: Build, train, and deploy machine learning models at scale.
   URL https://aws.amazon.com/sagemaker/
5. P. Ruf, M. Madan, C. Reich, D. Ould-Abdeslam, Demystifying mlops and presenting a recipe for the selection of open-source tools, Applied Sciences 11 (19) (2021) 8861.
6. K. Salama, J. Kazmierczak, D. Schut, Practitioners guide to mlops: A framework for continuous delivery and automation of machine learning, Google Could White paper (2021).
7. G. Fursin, H. Guillou, N. Essayan, Codereef: an open platform for portable mlops, reusable automation actions and reproducible benchmarking, arXiv preprint arXiv:2001.07935 (2020).
8. Y. Liu, Z. Ling, B. Huo, B. Wang, T. Chen, E. Mouine, Building a platform for machine learning operations from open source frameworks, IFAC-PapersOnLine 53 (5) (2020) 704–709, 3rd IFAC Workshop on Cyber-Physical & Human Systems CPHS 2020. doi:https:

//doi.org/10.1016/j.ifacol.2021.04.161. URL https://www.sciencedirect.com/science/article/pii/S2405896321003013

9. E. Peltonen, S. Dias, Linkedge: Open-sourced mlops integration with iot edge, in: Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum, 2023, pp. 67–76.
10. S. Wazir, G. S. Kashyap, P. Saxena, Mlops: A review, arXiv preprint arXiv:2308.10908 (2023).

Antonio M. Burgueño-Romero is a PhD candidate in the Department of Languages and Computer Sciences at the University of Málaga, Spain. His research interests include remote sensing, artificial intelligence, and MLOps. Burgueño received his MSc in Software Engineering and Artificial Intelligence from the University of Málaga. He is a member of the Khaos Research Group. Contact him at ambrbr@uma.es.

Antonio Benítez-Hidalgo is a PhD researcher in the Department of Languages and Computer Sciences at the University of Málaga, Spain. His research interests include large-scale data processing and big data management. Benítez received his PhD in semantics in big data analytics from the University of Málaga. He is a member of the Khaos Research Group. Contact him at antonio.b@uma.es

Cristóbal Barba-González is an assistant professor at the University of Málaga in Spain. His research interests include big data analysis, machine learning algorithms, and streaming processes. Barba received his PhD in multi-objective optimization with metaheuristics from the University of Málaga. He is a member of the Khaos Research Group. Contact him at cbarba@uma.es.

José F. Aldana-Montes is a full professor at the University of Málaga in Spain. His research interests include semantic middleware, semantic web, and big data analysis. Aldana received his professor position in computer science from the University of Málaga. He is the principal investigator of the Khaos Research Group. Contact him at jfaldana@uma.es.