END-TO-END MLOPS INTEGRATION: A CASE STUDY WITH ISS TELEMETRY DATA

Henrik S. Steude [1], **Christian Geier** [2], **Lukas Moddemann** [1], **Martin Creutzenberg** [2], **Jann Pfeifer** [3], **Samo Turk** [4], **and Oliver Niggemann** [1]
[1]Helmut Schmidt University, Hamburg, Germany, firstname.lastname@hsu-hh.de
[2]Just Add AI, Bremen, Germany, firstname.lastname@justadd.ai
[3]Airbus Defence & Space, Bremen, Germany, firstname.lastname@airbus.com
[4] Lufthansa Technik, Hamburg, Germany, firstname.lastname@lht.dlh.de

ABSTRACT

Kubeflow integrates a suite of powerful tools for Machine Learning (ML) software development and deployment, typically showcased independently. In this study, we integrate these tools within an end-to-end workflow, a perspective not extensively explored previously. Our case study on anomaly detection using telemetry data from the International Space Station (ISS) investigates the integration of various tools—Dask, Katib, PyTorch Operator, and KServe—into a single Kubeflow Pipelines (KFP) workflow. This investigation reveals both the strengths and limitations of such integration in a real-world context. The insights gained from our study provide a comprehensive blueprint for practitioners and contribute valuable feedback for the open source community developing Kubeflow.

Keywords MLOps · Kubeflow · ISS · Anomaly Detection

1 Introduction

MLOps, a fusion of Machine Learning (ML) and operations [10], has been an established concept for some years now. Especially since Sculley et al. [20] highlighted the complexity of operating ML applications in 2015. As a result, numerous new MLOps tools have been developed in recent years [14]. In parallel containerization of software gained traction as a popular way of ensuring compute workload resource allocation and user isolation. Kubernetes is one of the most popular container orchestration platforms providing abstraction to underlying infrastructure [2]. Among MLOps tools, Kubeflow emerged as a notable Kubernetes-based MLOps environment. It facilitates the integration of multiple ML tools and has gained attention in the community, with notable recognition including its incorporation in the CNCF Incubator [3].

Kubeflow is designed to support the entire lifecycle of ML projects, encompassing everything from data exploration to model development and retraining. Within this framework, Kubeflow integrates a variety of tools for distinct phases of the ML workflow.

However, existing documentations often focus more on the standalone functionalities of these tools, rather than exploring their combined potential in a cohesive workflow [7, 12, 11].

In response to this, our study aims to examine the practical application of these tools in an automated end-to-end workflow. Specifically, our research questions are as follows:

- Can Kubeflow's MLOps tools (e.g., Katib, PyTorch Operator, KServe) be effectively integrated into a pipeline for training, deploying, and re-training deep Neural Networks (NN) in an automated manner?
- What challenges exist in such pipelines and how can they be tackled?

To answer these questions, we conduct a case study based on a real-world application. The case study involves training and deploying an anomaly detection model for a subsystem of the International Space Station's (ISS) Columbus module. This use case presents an ideal scenario for our study due to the complexity of the data, which is more intricate than in most standard examples for ML tools. While MLOps spans a broad spectrum, including non-technical aspects such as Change Management and Teamwork, our study is primarily centered on the technical aspects.

Through this research, we contribute to the field of MLOps in two primary ways. Firstly, we have implemented an ML-workflow that showcases the integration of multiple tools within a single Kubeflow Pipelines (KFP) framework. The source code of our implementation is available on GitHub[1], offering a resource for practitioners who wish to explore similar applications or build upon our work. Secondly, we provide a detailed analysis of our case study, highlighting the functionalities that Kubeflow efficiently supports and identifies areas where further development could be beneficial. Our contributions to the open-source community, including reporting issues and submitting fixes, are part of our effort to refine these tools for enhanced usability and functionality.

2 Related Work

This section aims to situate our study in the broader context of current scientific research, offering an overview of the latest MLOps tools and reviewing recent studies that have employed case studies in MLOps.

State-of-the-art MLOps tools Recent publications compared Kubeflow with other tools for MLOps and evaluated them based on their ability to handle different phases of the MLOps process. E.g. Ruf et al. [19] divided the end-to-end MLOps process into five phases: *Data Preprocessing*, *Model Training*, *Model Management*, *Model Deployment*, and *Operations and Monitoring*. They evaluated 26 different tools for MLOps based on their ability to handle these phases. Among the evaluated tools, only three offered features for all five phases: Kubeflow, Polyaxon, and TFX. Two other tools offered features for four of the five phases: ZenML (excluding *Operations and Monitoring*) and Flyte (excluding *Data Preprocessing*). Köhler et al. [15] evaluated 30 different tools for MLOps based on three selection criteria: *(i)* the tool must be end-to-end, *(ii)* the tool must be marketed as Kubernetes native, and *(iii)* the tool must be open-source. Among the evaluated tools, Kubeflow, Polyaxon, and Pachyderm were the only ones that fulfill all three criteria.

Further, GitHub stars can be used as a measure of popularity for open-source software and the strength of their community [1]. See Table 1 for an overview of GitHub stars,

[1] https://github.com/hsteude/code-ml4cps-paper

as of January 2024, for the MLOps tools mentioned above. Considering this metric,

MLOps Tool	Number of GitHub Stars
Kubeflow	13.3k
Pachyderm	6k
Flyte	4.3k
Polyaxon	3.4k
ZenML	3.4k
TFX	2k

Table 1: Popularity of MLOps Tools based on GitHub Stars

Kubeflow appears to be the most popular among these tools and stands out as one of the few capable of handling the entire MLOps process end-to-end.

Case studies of MLOps in different disciplines Kubeflow has found application across diverse research domains, serving as a pivotal tool in bioinformatics for achieving rapid scaling through containerization [25]. Furthermore, researchers have utilized Kubeflow to establish automated ML workflows tailored for a service-aware 5G network model [24]. The adoption of a cloud-native approach emerges as a strategic choice, facilitating seamless deployment of workloads, even on public cloud resources. A Kubeflow platform was also successfully implemented in the context of CERN, where it was employed for expeditious simulation of electromagnetic showers using generative deep learning techniques [8]. In the realm of ML use case applications, various papers describe MLOps workflows designed to enhance collaboration and negotiation dynamics between surgeons and patients [9]. Additionally, there are instances where ML pipelines deployed on the cloud have been instrumental in drug discovery processes [22].

To our best knowledge, there are no existing studies that investigate the integration of Kubeflow's tools into an end-to-end workflow. Before presenting our case study in detail in Section 4, we will introduce the corresponding tools in the next section.

3 Method

As motivated in Section 1, our study aims to examine the integration of key MLOps tools within Kubeflow using the KFP framework. In this section, we describe the tools provided by Kubeflow that we have utilized in our case study. The specific application and purpose of these tools in the context of our case study will be detailed in Section 4.

To position these tools within the general ML workflow, we align them with the stages of the Cross-Industry Standard Process for Data Mining (CRISP-DM) [21] (see Figure 2), a process model that continues to be widely accepted, even over two decades after its introduction [16]. It is important to note that these tools are ML method agnostic and are applicable across a range of methodologies including supervised, unsupervised, and reinforcement learning.

Kubeflow Notebooks: For the initial stages of CRISP-DM, such as business understanding and data understanding, Kubeflow Notebooks offer an interactive environment for exploratory data analysis. Various web-based development environments, such as JupyterLab [18], can be utilized to perform interactive and visual data analyses. These environments run within Kubernetes, allowing for scalability in terms of resources (CPU/GPU and memory requests and limits) and enabling programmatic control of other Kubeflow tools described subsequently.

Dask: Although Dask [4] is not a part of Kubeflow, it complements the other tools described in this section well, especially as Kubernetes is one of the possible backends for Dask. Dask enables distributed processing of large datasets and offers a programming interface closely resembling those of Pandas [23]. Thus, it is frequently used in the 'Data Preparation' phase of the CRISP-DM cycle.

Katib: In the 'Modeling' phase, Katib [7] serves as a Kubernetes-native solution for automated ML (also called *AutoML* in the Kubeflow user interface), and especially hyperparameter tuning. Katib supports a wide range of popular ML frameworks, including PyTorch, TensorFlow, and scikit-learn, and implements various optimization algorithms for hyperparameter tuning. It enables the parallel execution of extensive hyperparameter tuning experiments, utilizing Kubernetes as its backend.

Training Operators: For better support of the training phase of ML models, Kubeflow includes Training Operators [2] [12]. They facilitate various training methodologies, including data- and model-parallel training, which are crucial for handling large NNs and extensive datasets, particularly important when dealing with foundation models. These Training Operators can also be integrated into Katib experiments.

KServe: In the 'Deployment' stage of CRISP-DM (not to confuse with the Kubernetes concept of *Deployments*), KServe [11] significantly streamlines the model deployment process. As a serverless[3] framework, it reduces deployment complexities for developers and provides optimized, high-performance inference capabilities. KServe includes features such as auto-scaling and built-in logging and monitoring, facilitating efficient model serving in production environments.

Kubeflow Pipelines (KFP): As a key component in Kubeflow, KFP [13] links various stages of the CRISP-DM cycle. It provides a framework for building, deploying, and managing scalable ML workflows. KFP executes distinct processing steps, known as pipeline components, arranged in a Directed Acyclic Graph (DAG) to ensure efficient data flow and adherence to workflow dependencies. Components within KFP can output data as either parameters for small datasets or as artifacts for larger files. KFP also tracks the data generated, orchestrating its flow between components and keeping records of each pipeline run and its artifacts.

In our case study, we focus on the practical application of a KFP setup, orchestrating the range of tools outlined in this chapter. The next section will delve into the specifics of this implementation, detailing the experiment and elucidating its results.

4 Results

Within this section we start with a description of the use case itself, followed by an outline of the specific steps of the ML pipeline developed for this purpose. The focus is on illustrating the integration and automation of the tools, as discussed in Section 3, within the framework of a Kubeflow pipeline.

[2] In Kubernetes, an 'Operator' is a method of packaging, deploying, and managing a Kubernetes application. Leveraging the built-in Kubernetes extension capacity, 'custom resources', Operators describe application-specific operational knowledge for managing the application's entire lifecycle. This encapsulation of operational procedures facilitates automation of routine tasks like deployment, scaling, failure recovery and configuration changes, thereby enhancing system reliability and efficiency.

[3] Serverless computing is an execution model where developers don't have to be concerned with capacity planning, management and scaling of their applications.

4.1 Use case

This case study is motivated by the need to identify anomalies in the telemetry data of the ISS, particularly within the Columbus module. Launched into orbit in February 2008 as a key contribution from the European Space Agency (ESA) with Airbus as prime contractor, the Columbus module offers a unique microgravity environment for a wide range of research activities [5]. It is continuously monitored and controlled by the Columbus Control Center (COL-CC) at the German Aerospace Center (DLR) in Oberpfaffenhofen, Germany.

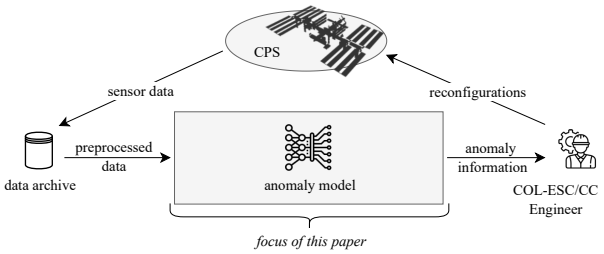

Figure 1: High-level overview of the information flow in the ISS diagnosis system.

From the business perspective, the primary focus of this case study is on the Environmental Control and Life Support System (ECLSS) of the Columbus module. ECLSS is crucial for maintaining a habitable environment within the module by regulating air circulation, temperature, and humidity. The goal of the project is to support the engineers at the Airbus-operated Columbus Engineering Support Center (COL-ESC) in detecting anomalies within the Columbus module's telemetry data stream using ML. This approach aims to enhance the efficiency of their daily operations and aid in the identification of hard-to-detect anomalous conditions. Examples of the varying nature and root causes of these anomalies can be found in the five-year operation report for the Columbus ECLSS subsystem [17].

Figure 1 depicts the information flow in the ISS anomaly detection use case. The telemetry data is transmitted to the ground stations, where the data is distributed, calibrated and archived. This preprocessed data is subsequently used for the anomaly detection model, both for training on historical data and inference on the incoming telemetry stream. For every anomaly detected by the system, information about time ranges and relevant subsystems are forwarded to the engineers in the COL-ESC. Following their investigation, they can decide to initiate countermeasures, such as suggesting reconfigurations of the system.

Columbus' telemetry data consist of thousands of distinct parameters, collected at a minimum frequency of one Hertz, streamed to Earth and archived over many years. A subset of this data has been labeled for the presence of anomalies by the responsible engineers, providing a crucial dataset for analysis.

From a technical perspective, this use case entails several typical ML project requirements: *(i)* exploration and preprocessing of large datasets, *(ii)* training of deep neural networks including hyperparameter tuning, *(iii)* deployment for real-time inference on streaming data, and *(iv)* regular retraining to accommodate data drift. In line with MLOps best practices, this project aims for a high degree of automation and full reproducibility.

4.2 ML-Pipeline

The pipeline implemented for this study encompasses several stages, structured to address distinct aspects of the ML workflow. These stages are: *(i)* initial data preparation, *(ii)* parallelized preprocessing, *(iii)* segmentation of telemetry data into training, development, and testing datasets, *(iv)* hyperparameter tuning, *(v)* model training, *(vi)* model evaluation, and *(vii)* model deployment. Figure 2 visualizes the pipeline as realized in KFP at a high level aligning the steps with the CRISP-DM cycle. The complete DAG and the associated source code[4] are available in our GitHub repository[5].

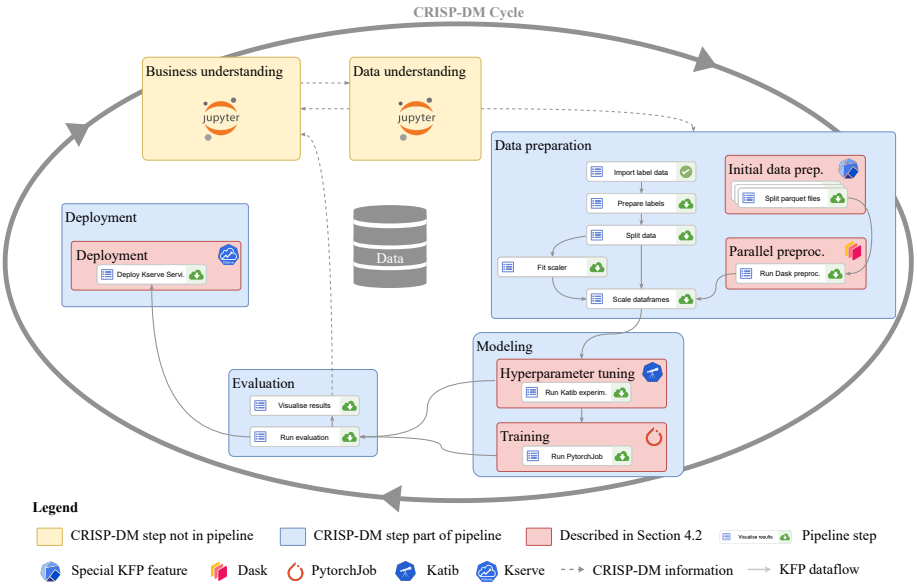

Figure 2: High-level overview of the ML pipeline implemented within KFP. This schematic illustrates the structured workflow aligned with the CRISP-DM process stages, integrating various tasks such as data preparation, modeling, hyperparameter tuning, and deployment. Key stages that incorporate additional Kubeflow tools are highlighted with red borders.

In the following, we will focus on the components highlighted in red in Figure 2, which are important to our research questions. These components demonstrate the integration of additional tools, namely Daks, Katib, PytorchJob and Kserve.

Initial Data Preparation The ECLSS subsystem's telemetry data were provided as annual Parquet files, each containing more than 1,000 time series sampled at 1Hz. Due to their large size, conventional data processing tools such as Pandas were inadequate. Additionally, the use of Dask for parallel processing necessitated the division of data into smaller segments. A KFP component was developed for this purpose, systematically dividing large files into smaller, processable units. Using this component, all the yearly archives files are split in a parallel for loop implemented using the KFP SDK.

[4]The source code uses the KFP SDK v2 and was tested on KFP backend version 2.0.0
[5]https://github.com/hsteude/code-ml4cps-paper

Parallel Preprocessing For preprocessing, Dask was employed to harness the full computational capacity of the Kubernetes cluster. The establishment of a Dask cluster, facilitated by the Dask Kubernetes operator and related Python package, enabled parallel data processing tasks within the pipeline. The cluster, composed of multiple worker pods[6], was dynamically assembled and subsequently dismantled after task completion, optimizing resource utilization.

Hyperparameter tuning For the hyperparameter tuning process, following the preparation of training, development, and testing datasets, we utilized Katib. This process involved utilizing the Kubernetes Python SDK for configuring and launching Katib training sessions within the pipeline. A key challenge encountered was the integration of Katib with KFP's data management system. Unlike other stages in the pipeline, where input/output operations are seamlessly managed by KFP as artifacts with data stored in object storage, Katib trials require a different approach for data access. To circumvent this limitation, we implemented custom code to handle data retrieval directly from object storage using the s3fs library for each trial.

Training Optimized parameters obtained in hyperparameter tuning process were passed to the PyTorch training job component. This stage required parallel training across multiple GPUs and nodes within the Kubernetes cluster, utilizing the Distributed Data Parallel (DDP) algorithm. Similar to the challenges faced during the hyperparameter tuning phase, the training job also encountered I/O challenges due to the worker pods not being managed by KFP. We tackled this as in the Katib job by reading from and writing data to object storage.

Deployment Model deployment, the final step in the pipeline, was facilitated by KServe. KServe utilizes model training artifacts (e.g., scaler, model, etc.) to establish a REST endpoint for inference. Similar to Katib and training job operators, KServe is not directly integrated into pipelines, thus it cannot leverage seamless artifact handling. Further, users are required to explicitly implement the configuration (i.e., manifests) and code to deploy the inference service from the pipeline. Finally, while permissions for most services/steps in the pipeline are pre-configured, an administrator must specifically grant users permission to deploy models using KServe.

Model evaluation Though the primary focus of this study is to demonstrate the integration of various Kubeflow tools, we briefly discuss the outcomes of the model evaluation to illustrate the applicability of the implemented pipeline here.

A KFP component was developed for model evaluation. This component loads the trained model and performs inference on the test set, which was neither used for training nor validation. To assess the model's performance, we adopted the *Composite F-Score*, as introduced in [6]. This score, designed for anomaly detection in multivariate time series, optimizes a balance between time-wise precision and event-wise recall, under the assumption that an ideal anomaly detector should identify at least one point per anomaly event while minimizing false positives.

Additionally, scatter plots were generated to visualize the temporal progression of the model's likelihood scores against the true anomaly labels (see Figure 3). KFP offers the capability to display these metrics and visualizations within its pipeline UI, enabling comparison across different pipeline runs with varying input parameters. For

[6] A pod is the smallest unit of computing on Kubernetes and is composed of a one or more containers sharing storage and network.

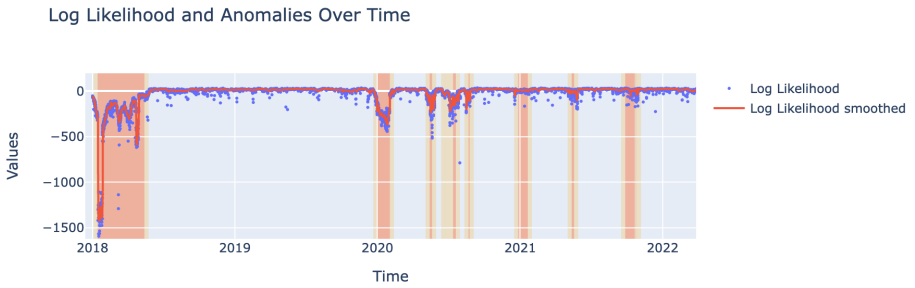

Figure 3: Log likelihood (blue dots) of anomaly detection over time, with a smoothened curve (red line) for trend visualization. Red shaded areas represent periods with labeled anomalies. The yellow shaded areas denote segments surrounding the anomalies that are exclusive to the test set and have been labeled as non-anomalous.

our experimentation, we deployed the pipeline on a Kubernetes cluster comprising three nodes, each equipped with 128 CPU cores and 500GB of memory, alongside a total of 4 NVIDIA A30 GPUs. The input parquet files amounted to 7.5GB. In our case study, the implemented anomaly detection model achieved a *Composite F-Score* of 0.77, which is considered satisfactory given the complexity of the task.

5 Discussion

In this paper, we have explored the orchestration of multiple tools within Kubeflow making use of the KFP framework for tying several tools and other frameworks together and demonstrated that such integration is indeed feasible. This section aims to discuss the aspects of our implementation that were particularly successful, as well as those that presented challenges, necessitating workarounds.

The points raised in this section are a curated selection, chosen to emphasize on those aspects we deem most significant for users considering Kubeflow and for contributors involved in its ongoing development. The highlighted strengths and challenges are based on our direct experiences, serving both as a guide for prospective adopters and as constructive feedback to inform continuous improvement of the platform.

In the course of our work with the Kubeflow framework, we identified several key strengths that facilitated the orchestration of various MLOps tools:

- **Compute Resource Sharing:** Kubeflow is well-suited for sharing Kubernetes cluster resources among teams, as pods release all resources upon job completion. This is also true for notebooks, which scale resources based on utilization between user-defined requests and limits, and also holds for Katib experiments and training jobs.
- **Compute Resource Merging** The training operators facilitate Neural Network training on multiple GPUs and compute nodes simultaneously, allowing not only efficient resource sharing but also the consolidation of these resources for intensive tasks.
- **Unified Environment for the Entire Process:** Theoretically, data scientists do not need to leave the Kubeflow environment to complete the entire described process, with the potential to work entirely within a browser-based interface.

- **Caching of Task Results:** KFP allows for caching, which is particularly useful for long-running processes at the pipeline's start. Changes towards the pipeline's end, like evaluation or result visualization, do not necessitate re-running these processes, ensuring reproducibility while managing artifacts from each step.
- **Effective Collaboration:** Inviting team members into one's Kubeflow namespace simplifies collaboration on common pipelines or notebooks.
- **High Level of Automation:** As the pipeline encapsulates the entire end-to-end process of model creation, it facilitates the fully automated generation and deployment of new model versions. Such retraining can be triggered by the availability of new data, detection of model drift, or after a predefined period of time.

Despite the successes, our implementation faced several challenges:

- **Need for More Comprehensive Documentation:** Compared to other ML frameworks, Kubeflow's documentation is less developed and would benefit significantly from more end-to-end examples.
- **Pipeline I/O Limitations:** While KFP simplifies pipeline development and usage, its I/O tools are not fully compatible with PyTorch jobs, Katib experiments or KServe. Integrating those tools into the KFP workflow required implementing custom workarounds. While writing custom code for data access is not necessarily a problem, it does require a deeper understanding of the underlying tools and their APIs.
- **Limitations of the Open-Source Server-Side Kubeflow Pipelines:** The server-side component of Kubeflow Pipelines is offered in both open-source and proprietary forms. The open-source variant has not yet fully implemented certain control flow features, such as advanced options for if-conditions and parallel for loops, which are available in the proprietary versions.
- **Ongoing Development and Emerging Issues:** During our case study implementation, we encountered features that were only available in SDK v1, which has since been replaced by SDK v2. This highlights Kubeflow's active development stage, indicating that users should be prepared for minor bugs and evolving features.

To summarize, our case study has underscored Kubeflow's potential as a robust platform for managing the complexities of MLOps tasks, thanks to its resource management capabilities, cohesive environment, and experiment tracking features. Nevertheless, the challenges we have outlined, particularly concerning documentation and tool integration, suggest that there is still room for refinement to fully realize this potential.

6 Conclusion

In this paper, we have explored the feasibility of integrating and automating a suite of MLOps tools within the Kubeflow framework, with a focus on a case study involving anomaly detection in telemetry data from the ISS Columbus module. Our research has successfully demonstrated that such integration is achievable, providing insights into the practical application of Kubeflow for complex MLOps tasks.

While we have identified KFP as an effective platform for managing MLOps tasks, we also encountered challenges, particularly in terms of documentation and tool integra-

tion. These challenges highlight that Kubeflow, although promising, requires further development and refinement to fully realize its potential.

In summary, our study shows that Kubeflow is a robust tool for MLOps, particularly distinguished by its resource management capabilities, provisioning of a cohesive environment for the entire ML process, and strong community support. Despite the challenges observed, our work contributes to the MLOps community by demonstrating a pathway for efficiently managing complex ML workflows.

For future work, we believe that focusing on improving Kubeflow's documentation, addressing its pipeline I/O limitations, and enhancing tool compatibility are among the most important aspects.

Acknowledgments

This research – as part of the *(K)ISS* project – is funded by *dtec.bw – Digitalization and Technology Research Center of the Bundeswehr*. dtec.bw is funded by the European Union through *NextGenerationEU*.

References

[1] H. S. Borges, A. Hora, and M. Valente. Understanding the factors that impact the popularity of GitHub repositories, 2016.

[2] E. Casalicchio and S. Iannucci. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, 32(17):e5668, 2020.

[3] Cloud Native Computing Foundation. Kubeflow brings MLOps to the CNCF incubator. `https://www.cncf.io/blog/2023/07/25/kubeflow-brings-mlops-to-the-cncf-incubator/`, 2023. Accessed: January 11, 2024.

[4] Dask Development Team. Dask: Parallel computing in python. `https://github.com/dask/dask`, 2023. Accessed: January 11, 2024.

[5] eoPortal Authors. ISS: Columbus module. `https://www.eoportal.org/satellite-missions/iss-columbus`, 2012. Accessed: January 20, 2024.

[6] A. Garg, W. Zhang, J. Samaran, R. Savitha, and C.-S. Foo. An evaluation of anomaly detection and diagnosis in multivariate time series. *IEEE Trans Neural Netw Learn Syst*, PP, Aug. 2022.

[7] J. George, C. Gao, R. Liu, H. G. Liu, Y. Tang, R. Pydipaty, and A. K. Saha. A scalable and cloud-native hyperparameter tuning system, 2020.

[8] D. Golubovic and R. Rocha. Training and serving ml workloads with kubeflow at cern. In *EPJ Web of Conferences*, volume 251, page 02067. EDP Sciences, 2021.

[9] T. Granlund, V. Stirbu, and T. Mikkonen. Towards regulatory-compliant MLOps: Oravizio's journey from a machine learning experiment to a deployed certified medical product. *SN computer Science*, 2(5):342, 2021.

[10] D. Kreuzberger, N. Kühl, and S. Hirschl. Machine learning operations (MLOps)): Overview, definition, and architecture. *IEEE Access*, 11:31866–31879, 2023.

[11] KServe Authors. Kserve. `https://github.com/kserve/kserve`, 2023. Accessed: January 11, 2024.

[12] Kubeflow Authors. Training operator. `https://github.com/kubeflow/training-operator`, 2023. Accessed: January 11, 2024.

[13] Kubeflow Pipelines Authors. Kfp. `https://www.kubeflow.org/docs/components/pipelines/`, 2023. Accessed: January 11, 2024.

[14] I. Kumara, R. Arts, D. Di Nucci, W. J. Van Den Heuvel, and D. A. Tamburri. Requirements and reference architecture for MLOps:Insights from industry. *Authorea Preprints*, Nov. 2023.

[15] A. Köhler. Evaluation of MLOps tools for kubernetes: A rudimentary comparison between open source Kubeflow, Pachyderm, and Polyaxon. `https://www.diva-portal.org/smash/get/diva2:1711840/FULLTEXT01.pdf`, October 2022. IT 22 119.

[16] F. Martinez-Plumed, L. Contreras-Ochando, C. Ferri, J. Hernandez-Orallo, M. Kull, N. Lachiche, M. J. Ramirez-Quintana, and P. Flach. CRISP-DM twenty years later: From data mining processes to data science trajectories. *IEEE Trans. Knowl. Data Eng.*, 33(8):3048–3061, Aug. 2021.

[17] P. Parodi, Z. Szigetvari, R. Muller, and A. Quaglia. Columbus ECLSS: five years of operations and lessons learned. In *43rd International Conference on Environmental Systems*, 2013.

[18] J. M. Perkel. Why Jupyter is data scientists' computational notebook of choice. *Nature*, 563(7729):145–146, Nov. 2018.

[19] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam. Demystifying MLOps and presenting a recipe for the selection of open-source tools. *Applied Sciences*, 11(19):8861, 2021. Accessed: Januar 8, 2024.

[20] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. *Adv. Neural Inf. Process. Syst.*, 2015.

[21] C. Shearer. The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), 2000.

[22] O. Spjuth, J. Frid, and A. Hellander. The machine learning life cycle and the cloud: implications for drug discovery. *Expert opinion on drug discovery*, 16(9):1071–1079, 2021.

[23] The pandas development team. pandas-dev/pandas: Pandas, Feb. 2020.

[24] T. Tsourdinis, I. Chatzistefanidis, N. Makris, and T. Korakis. AI-driven service-aware real-time slicing for beyond 5G networks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2022.

[25] D. Y. Yuan and T. Wildish. Bioinformatics application with kubeflow for batch processing in clouds. In *International Conference on High Performance Computing*, pages 355–367. Springer, 2020.