Future Generation Computer Systems 163 (2025) 107499

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Big Data-driven MLOps workflow for annual high-resolution land cover classification models

Antonio M. Burgueño-Romero *, Cristóbal Barba-González, José F. Aldana-Montes

KHAOS Research group, ITIS Software, Universidad de Málaga, Málaga 29071, Spain

ARTICLE INFO

Dataset link: https://github.com/KhaosResearch/landcoverpy, https://github.com/KhaosResearch/mlops-landcoverpy, https://github.com/KhaosResearch/mlops-infra

Keywords:
MLOps
Land cover
Big Data
Kubernetes
Remote sensing

ABSTRACT

Developing an annual and global high-resolution land cover map is one of the most ambitious tasks in remote sensing, with increasing importance due to the continual rise in validated data and satellite imagery. The success of land cover classification models largely hinges on the data quality, coupled with the application of Big Data techniques and distributed computing. This is essential for efficiently processing the extensive volume of available satellite data. However, maintaining the lifecycle of several annual Machine Learning models presents a complex challenge. The rise of Machine Learning Operations offers an opportunity to automate the maintenance of these models, a feature particularly beneficial in systems that require generating new models each year alongside the continuous integration of validated data. This article details the development of an end-to-end MLOps workflow, meticulously integrating land cover classification models that employ Big Data strategies for processing large-scale, high-resolution spatial data. The workflow is designed within a Kubernetes environment, achieving on-demand auto-scaling, distributed computing, and load balancing. This integration demonstrates the practicality and efficiency of managing and deploying models that treat satellite imagery in an automated, scalable framework, thus marking a significant advancement in remote sensing and MLOps.

1. Introduction

The significant impact of satellite-derived information on scientific research is widely acknowledged. Initially, satellite data relied on expert interpretation, supported by statistical models derived from the data. However, with the advancements in Machine Learning (ML) over the past decades, experts are now engaged in validating and fine-tuning ML models. These models have revolutionised data analysis and interpretation, enabling more accurate and efficient satellite Big Data analysis. Integrating ML techniques has enhanced the capacity to uncover hidden patterns, make predictions, and gain insights from satellite-derived information, thus contributing to advancements across various scientific domains.

Remote sensing analysis for land cover (LC) classification uses remote sensing data, such as satellite imagery or aerial photographs, to classify and map different LC types and land use patterns over a large geographic area [1]. LC refers to the physical characteristics of the Earth's surface, including forests [2], agricultural fields [3], urban areas [4], water bodies [5], etc. While domain experts predominantly accomplished LC mapping in the past, contemporary approaches incorporate ML models as an integral part of the process [6].

ML has emerged as a powerful tool for extracting valuable insights and making predictions or classifications from complex data [7]. ML has witnessed significant advancements in recent years, leading to the development of sophisticated algorithms and techniques [8]. One area where ML has demonstrated remarkable success is Big Data analysis [9]. Big Data enables ML algorithms to uncover more fine-grained patterns and make more timely and accurate predictions [10]. Furthermore, enormous data presents significant challenges to ML, such as model scalability and distributed computing [11]. The success of ML models heavily relies on the quality and quantity of data, feature engineering, algorithm selection, model training, hyperparameter tuning, and deployment [12].

Moreover, the iterative nature of ML projects necessitates a robust and scalable framework to manage the entire ML lifecycle efficiently [13]. In this sense, to create the specifications of the processes for defining ML lifecycles efficiently, in the last decades, DevOps has appeared [14], which includes two main elements: firstly, the development, which means all the people related to the software, including programmers, testers, and quality assurance staff. Secondly, the operations represent the professionals who deploy software into production and manage the production infrastructure, including system administrators, database administrators, and network technicians [15].

* Corresponding author.
E-mail addresses: ambrbr@uma.es (A.M. Burgueño-Romero), cbarba@uma.es (C. Barba-González), jfaldana@uma.es (J.F. Aldana-Montes).

https://doi.org/10.1016/j.future.2024.107499
Received 10 January 2024; Received in revised form 5 June 2024; Accepted 25 August 2024
Available online 28 August 2024
0167-739X/© 2024 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

Machine Learning Operations (MLOps) [16] extends the widely adopted DevOps practices [17]. MLOps focuses on managing and operationalising ML models systematically and efficiently. By integrating the best practices from DevOps and ML, MLOps enables organisations to automate the ML lifecycle, improve team collaboration, ensure reproducibility, and accelerate time-to-market for ML-driven applications [18]. Besides, Artificial Intelligence (AI) for IT Operations (AIOps) [19] assists DevOps and MLOps in improving the quality and reliability of IT service offerings. AIOps employ algorithms and the maximum data available thanks to monitoring the infrastructures. Thus, AIOps help enhance service quality, raise engineering productivity, and reduce cost. Together, these three disciplines form a comprehensive framework for effectively managing the entire lifecycle of AI and ML projects, from development to deployment and maintenance. By leveraging the principles and practices of DevOps and MLOps within the broader context of AI Ops, organisations can drive innovation, enhance productivity, ensure reliability, and achieve operational excellence in the rapidly evolving world of AI.

MLOps is particularly crucial in the context of annual high-resolution LC classification models [20] due to the dynamic nature of LC and the continuous influx of new, validated data. Each year, changes in LC, influenced by natural phenomena and human activities, necessitate the updating and refinement of classification models. The annual cycle demands a systematic approach to incorporate new satellite imagery and ground truth data, ensuring models adapt to current realities. This maintenance of one LC model for each year allows us to detect LC changes and make historical classifications of the land, not only enhancing the accuracy of yearly models but also ensuring consistency and comparability across different periods. Integrating MLOps in LC classification thus represents a significant step towards more robust, reliable, and up-to-date environmental monitoring and analysis.

Traditionally, LC classification has relied on manual workflows, where the process involves a series of steps such as data preprocessing, feature extraction, model training, and validation, all done manually. This approach, while effective in certain contexts, is often labour-intensive, time-consuming, and prone to human error. Manual workflows also struggle to keep pace with the rapid influx of new data, making it challenging to update models in a timely manner. Additionally, the lack of automation can lead to inconsistencies in the data processing and model updating steps, which can impact the overall accuracy and reliability of the classification models.

In contrast, automated workflows leverage MLOps principles to streamline the entire lifecycle of LC classification models. Automated pipelines facilitate continuous integration and deployment of models, allowing for seamless updates and maintenance as new data becomes available. This automated approach not only reduces the time and effort required for model updates but also enhances the consistency and reproducibility of the results. By incorporating automated data ingestion, preprocessing, model training, and evaluation, the MLOps framework ensures that LC models remain up-to-date and accurately reflect current land cover conditions.

Some studies have started incorporating automated pipelines to LC models [21]. However, a significant research gap remains in defining a comprehensive environment that includes all the essential components. Consequently, there is a pressing need to explore the implementation of MLOps practices within the domain of LC modelling. Such an endeavour would facilitate the efficient development, deployment, monitoring, and maintenance of annual LC models.

Kubernetes [22] and microservices architecture are pivotal in Big Data and MLOps environments for efficiently managing ML models at scale [23]. By leveraging containerisation, Kubernetes enables the encapsulation of ML components and their dependencies, ensuring portability, consistency, and reproducibility across different environments. Containers offer lightweight isolation, making package, deploying, and scaling ML applications consistently easier [24].

The central aim of this study is to implement advanced MLOps methodologies, as outlined in the whitepaper [25], within the realm of LC analysis, mainly focusing on annual high-resolution LC classification models. Tackling high-resolution LC challenges, such as those involving 10 m spatial resolution Sentinel-2 imagery, necessitates addressing complex Big Data and scalability issues. We will employ open-source, state-of-the-art software tools in a Kubernetes environment to maintain several high-resolution LC models. This approach is designed to automate the models' lifecycle, significantly reducing manual intervention, enhancing reproducibility and uniformity, and providing auto-scaling functionality. The research aims to showcase the effectiveness of MLOps practices in managing extensive datasets and proficiently handling LC models. This initiative is expected to refine these models' development, deployment, and maintenance, guaranteeing their enhanced performance and scalability.

In pursuit of these objectives, we have developed an AI-as-a-service (AIaaS) solution based on a Big Data-oriented methodology, as previously introduced [26]. This methodology is employed for training annual LC models using year-specific validated coordinates, complemented by corresponding satellite imagery to capture terrain characteristics for that specific year accurately. This work manages these models within an MLOps environment deployed on a Kubernetes cluster. The whole process enables users to obtain precise terrain classification (outlined with a geoJSON) through a REST or gRPC API for any year since the availability of Sentinel-2 data. By implementing this approach, we provide a comprehensive and efficient solution that leverages advanced technologies and methodologies, facilitating seamless access to precise LC information for various applications.

This paper introduces a meticulously crafted MLOps workflow designed for high-resolution LC classification. This auto-scalable solution is adept at automatically maintaining annual models. It encompasses many functionalities, including data processing, model retraining, model registry, model serving, online experimentation, and model monitoring. Collectively, these features ensure that our system not only provides an updated version of the model accessible via REST or gRPC endpoints at any time but also continuously integrates newly validated data to enhance the model's accuracy.

The workflow has been implemented using open-source software and is made publicly available. This includes both the deployment of the framework in a Kubernetes cluster[1] and the necessary components for deploying the LC model within this environment[2] (model containerisation, retraining workflow, etc.). The methodology for training the base model, packed in a Python library,[3] were previously published in our earlier works [26].

The remainder of the paper is organised as follows. Section 2 includes the main related work. Section 3 details the different phases of the proposal workflow. Section 4 delves into the approaches and techniques used in our LC classification. Section 5 analyses the results obtained. Section 6 is dedicated to discuss the practical difficulties encountered during the implementation of our framework. Finally, conclusions are drawn, and future work is proposed in Section 7.

2. Literature review

Over the last few decades, the software industry has evolved from boxed product delivery to service-oriented releases, encompassing applications and services [27]. This trend has not bypassed ML model development, underscoring the importance of continuous software engineering practices [28]. Disciplines such as AIOps [27], DevOps, and MLOps have emerged to streamline the development and deployment of AI and ML solutions. AIOps focuses on the efficient and effective

[1] https://github.com/KhaosResearch/mlops-infra
[2] https://github.com/KhaosResearch/mlops-landcoverpy
[3] https://github.com/KhaosResearch/landcoverpy

(a) Deployed 10 meters LC using random forests. (b) Sentinel-2 10 meters true colour composite image. (c) State-of-the-art 10 meters LC using CNNs.

Fig. 1. 1(a), 1(b) and 1(c) present a comparative analysis of a specific region in Cintruénigo, Spain. While the LC classes may vary between the approaches, both demonstrate the presence of built-up areas (red) and cropland (pink). Notably, the state-of-the-art LC model exhibits characteristics of object-based classification despite the spatial resolution of 10 m. In contrast, the model using our methodology, leverages a pixel-based classification approach, enabling a higher level of detail in LC classification across any given area.

operation of services using AI and ML techniques [29,30], while DevOps integrates software development with IT operations, advocating for frequent releases and automation through continuous integration, delivery, and deployment (CI/CD) [31–33].

MLOps, gaining popularity in recent years [34], extends DevOps principles to ML, offering several advantages over traditional methodologies [35]. It involves orchestrating middleware and software components to realise key workflow steps: data gathering, transformation, continuous ML (re-)training, (re-)deployment, and output production/presentation [36]. However, automating and operationalising ML products remains a challenge, often leading to unmet expectations in ML projects [32]. This paper seeks to bridge these gaps, focusing on the application of MLOps in LC analysis, drawing from various research areas and practices [37–43].

The pursuit of an automatically generated, large-scale, and high spatial resolution annual LC map is a prominent topic in remote sensing. Traditional approaches have seen a gradual shift from manual methods to automated techniques that combine ML and human intervention. Despite advancements in AI, current LC maps like the CORINE land cover (CLC) product [44], the 500 m MODIS land cover Type product [45], and the 300 m European Space Agency Climate Change Initiative (CCI) land cover product [46] still require post-processing human intervention. An interesting development is the collaboration between Esri and Microsoft [47], which produced a global-scale LC map with a spatial resolution of 10 m (see Fig. 1(c)). However, this map, utilising a convolutional neural network (CNN), predominantly yields object-based results. This approach might not attain the pixel-level accuracy characteristic of pixel-based methods. An example of a LC classification methodology, which effectively handles Big Data (extensive areas while maintaining 10 m spatial resolution) and leverages distributed execution, is the one we previously introduced in [26] (see Fig. 1(a) for details). Fig. 1 provides a comparative analysis, comparing the current state-of-the-art in high-resolution LC mapping against the results achieved using our methodology.

The increasing availability of validated location data over multiple years [48,49] has spurred advancements in optimising training workflows for LC models, as seen in recent studies [50]. These works, however, primarily address the orchestration of workflows and fall short in incorporating a full range of MLOps practices, especially those crucial for the deployment and maintenance of models in multi-temporal datasets. This gap highlights the need for research that not only develops accurate LC models but also automates their deployment and management, ensuring their relevance and effectiveness over time.

In contrast, recent works such as [51–53] apply some MLOps best practices for map generation from Sentinel-2 Satellite Imagery. These efforts, however, are limited to smaller scales and do not outline a complete MLOps environment, often lacking the necessary code for infrastructure deployment. Our work addresses these gaps by presenting a comprehensive MLOps workflow integrating a Big Data-oriented LC methodology and providing transparent code for each workflow component.

3. MLOps workflow methodology

This section presents a comprehensive and robust workflow (see Fig. 2) encompassing all state-of-the-art MLOps components for any ML model. This workflow significantly reduces the need for human intervention to validate new training data while automating crucial steps such as retraining, versioning, production redeployment, real-time monitoring, and auto-scaling.

Overall, our workflow streamlines the implementation of MLOps practices, ensuring the efficient management and deployment of ML models while enhancing user experience by simplifying the prediction process and delivering accurate results.

It is worth noting that although we have seamlessly integrated the Big Data-ready LC methodology into our MLOps workflow, our established infrastructure is independent. Any other model type can also be integrated, provided that the retraining pipeline is customised and proper model containerisation is implemented.

The methodology presented stands out by leveraging independent open-source components, ensuring a flexible and modular MLOps workflow. This design differs from open-source frameworks like Kubeflow or paid cloud services like Azure Machine Learning; each infrastructure element can progress and improve autonomously. This independence allows for continuous evolution and the freedom to substitute any component with an alternative solution in case project directions shift or preferences change, a level of agility and customisation that proprietary or less modular systems cannot easily replicate.

Each step of the workflow is described as follows:

1. ***Model registry***. The versions of the model are stored in a model registry, a component designed to centralise model storage, versioning, and metadata storage. It aims to ensure reproducibility, track progress between versions, and facilitate metadata management. The model registry connects to an object storage, housing artefacts such as models, code, and datasets, and a relational database containing all necessary metadata, including

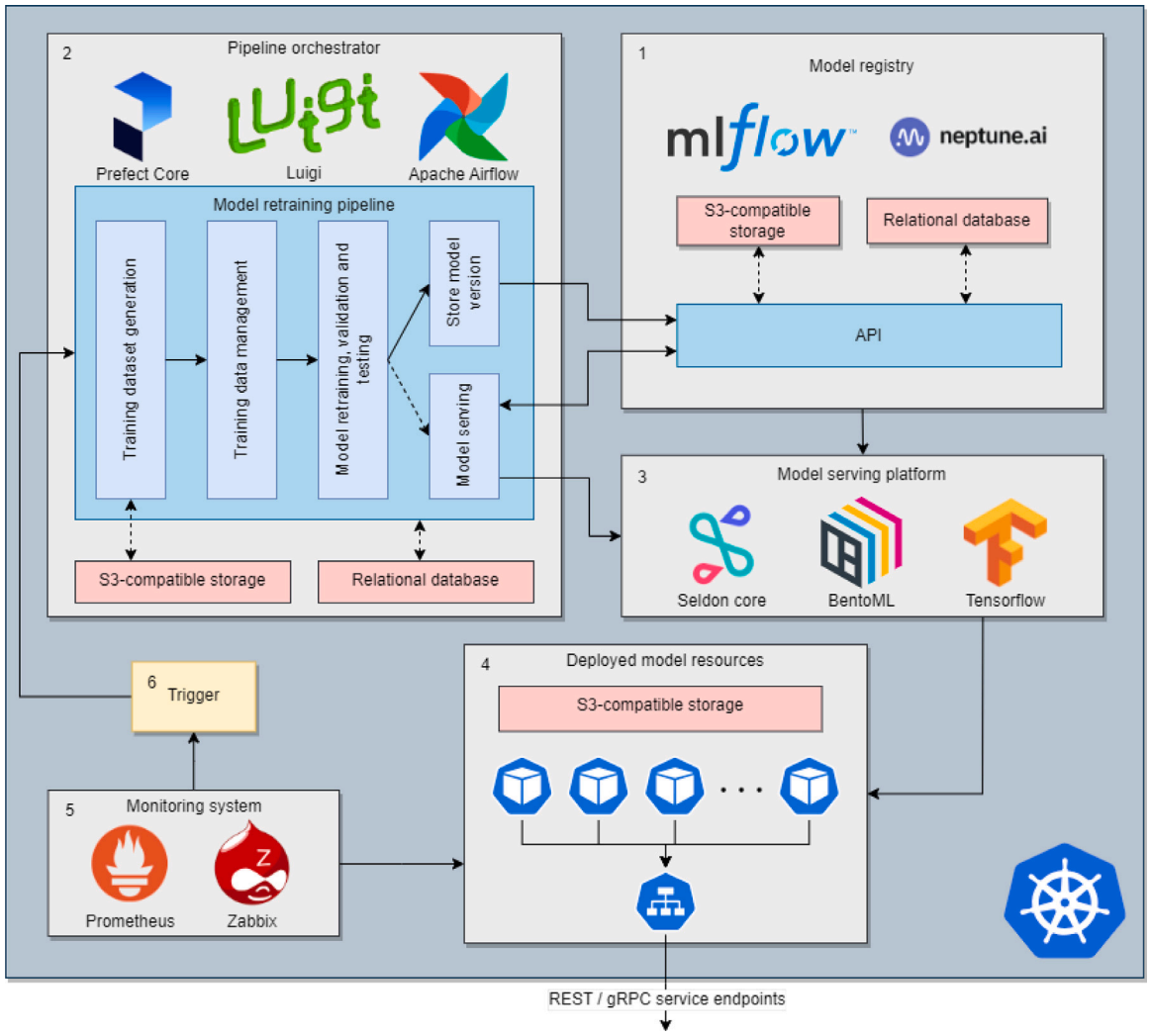


Fig. 2. Proposed workflow for automating the lifecycle management of any ML model. The workflow contains six steps. Each step is accomplished by a container representing a component (grey). All the components are deployed within a Kubernetes environment. Furthermore, the logo of popular open-source software that implements the environment is incorporated into each component. Light red containers indicate either object storage or databases.

dates, execution times, IDs, and other relevant information. Although we used MLflow [54] as a model registry due to its strong community support and extensive documentation, which facilitated smoother implementation and troubleshooting, neptune.ai [55] is an excellent open-source alternative (see step 1 in Fig. 2).

2. ***Pipeline orchestrator***. It is essential to have a pipeline orchestrator to retrain the model when necessary. This component enables the definition of task workflows and their automation, thereby reducing or eliminating the need for human intervention. Additionally, it ensures reproducibility, dependency management, scalability, and parallel processing. One of the critical aspects of a pipeline orchestrator is its ability to store the defined pipelines and executors within containers. This ensures that the pipeline can be executed in any available runtime environment, guaranteeing its functionality regardless of the specific execution environment being utilised. Furthermore, the pipeline orchestrator can efficiently manage model deployment as needed. When retrieving the deployed model version or any related metadata, the model registry is of great assistance, as it provides an API that simplifies access to the required data, abstracting object storages, directories, and other complexities unrelated to the model itself. Although the implementation of the retraining pipeline depends on the specific use case, it typically comprises the same logical components [56], including data ingestion, data preparation for model retraining, retraining and validation of the new version, storing the updated model in the model registry, and serving. The pipeline orchestrator component typically connects to a relational database containing tables for task executions, versioning, task state tracking, and other relevant information. Although we used Prefect Core [57] as pipeline orchestrator due to its modern Python-based API, which allows for more readable and maintainable code as well as greater flexibility in handling dynamic workflows, Luigi [58] and Apache Airflow [59] are good open-source alternatives (see step 2 in Fig. 2).
3. ***Model serving platform***. Once the best version of the model has been determined, it needs to be served to end users. This involves deploying an API that provides easy access to the model and prepares it for real-time predictions in production environments. The goal of a model-serving platform is to streamline the process of deploying a model in production. Additionally, it offers scalable capabilities to accommodate varying user loads and support the deployment of multiple model versions concurrently, allowing for A/B testing and easy rollback to previous versions if needed. Moreover, the platform should facilitate the collection of metrics and statistics on model performance, usage,

and potential issues in production, enabling continuous monitoring and improvement of the deployed model's effectiveness and reliability. Finally, it also facilitates the process of building a container that provides an API with several methods capable of making predictions using any model. Although we used Seldon Core [60] as model serving due to its robust support for Kubernetes, which ensures scalable and reliable deployments, and its ability to support complex inference graphs and advanced metrics tracking, BentoML [61] and Tensorflow [62] are good open-source alternatives (see step 3 in Fig. 2).

4. ***Deployed model resources***. Regardless of the platform, a deployed model consists of containers with which the user communicates through requests (usually via a REST/gRPC API) over one or several defined ports. Additionally, it typically defines additional ports for accessing real-time metrics, which export at each particular time or when a user request arrives. Typically, multiple replicas of the model are deployed to distribute user requests and enhance performance. User requests are directed to a service that exposes all model instances to an external network (see step 4 in Fig. 2).
5. ***Monitoring system***. It is responsible for scrapping real-time metrics from the models through a designated port. However, an additional component capable of providing real-time monitoring and alerting/anomaly detection is required for effective monitoring. This component will observe the metrics exported by each model and promptly raise alerts upon detecting any anomalies in their performance. It is essential that experts set thresholds over specific metrics for alarms. Ultimately, the responsibility of triggering the complete re-execution of the workflow will likely fall on this component. In addition to the metrics used to trigger retraining, monitoring metrics related to model performance is essential. These metrics provide insights into the system's overall health and indicate whether everything functions as expected. Although we used Prometheus [63] for monitoring due to its powerful query language, PromQL, and its seamless integration with Kubernetes, which allows for efficient monitoring of containerised applications, Zabbix [64] is an excellent open-source alternative (see step 5 in Fig. 2).
6. ***Trigger***. One of the most crucial aspects of the MLOps framework is determining the trigger point for initiating the entire retraining, registration, and redeployment process if needed. Automating this trigger can be achieved in various ways, but it should be noted that each use case is unique, as data availability differs across scenarios. The simplest option is to schedule the trigger regularly or when a certain quantity of data becomes available. This approach works well for use cases where new data is obtained periodically. Another approach involves real-time monitoring. Typically, metrics on model performance are available in real-time (usually when a user request is fulfilled). As mentioned, monitoring systems often include an alarm system that notifies developers when the model's performance deviates from expectations. The entire process can be executed if sufficient data is available when one or a combination of alarms is triggered (see step 6 in Fig. 2).

It is essential to consider that this entire framework needs to integrate the standard continuous integration/continuous deployment (CI/CD) techniques of DevOps to function correctly. The framework includes CD techniques when performing model registry, model serving, etc. However, the CI techniques are not as straightforward. It integrates code from a repository that provides automated tests and automated workflows to update the containers used in the MLOps environment, such as the retraining pipeline or the model serving wrapper.

Using Kubernetes as the environment for deploying all framework components allows for easy management of model replicas, ensuring smooth horizontal scaling to handle varying user loads and demands. Additionally, Kubernetes provides robust features for automated deployment, monitoring, and self-healing capabilities, ensuring the system remains resilient and available despite failures. This approach, however, can be applied effectively on various cloud platforms or container orchestration systems, making it adaptable to diverse environments.

4. Land cover methodology

Although the framework presented in this paper has been designed to apply to any ML model, it is theoretically valid for diverse applications. However, in this research article, we have specifically developed it into an actual use case involving an LC model.

The initial step in any ML project involves obtaining a functional baseline model. In previous works, we presented a robust methodology to train annual LC models (see Fig. 1(a)) using high-resolution satellite data, addressing the challenges Big Data poses.

To develop the initial stable version of the LC model, 15 features derived from Sentinel-2 [66] and ASTER [67] data have been utilised, incorporating over 40,000 manually validated locations across nine distinct classes, achieving a pixel-level classification accuracy of over 75%. This detailed methodology encompasses the computation of monthly composites during different seasons, data harmonisation, various machine-learning techniques, and distributed processing at the Sentinel-2 tile level. The generation of the tabular dataset for training an annual model with our validated data involved processing over 10TB of raw satellite imagery. This extensive data comprised more than 6,000 Sentinel products and 1,200 from ASTER. Given the sheer volume of data to be processed, applying Big Data techniques and distributed computing was imperative. All the code created for the LC is distributed under a free and open-source license.[4]

To create an AIaaS solution that offers a user-friendly interface for LC prediction, we further enhanced the model's prediction capability to accept a geoJSON input specifying the target area of interest (AOI). The model then returns a link to download a raster file containing the classified AOI, providing users with convenient access to the predicted LC information.

In remote sensing, it is expected to receive a continuous stream of validated data spanning current and past years. Integrating new data requires retraining and validating all annual models to maintain their accuracy and relevance. This ongoing need to update and manage multiple models highlights an automated system's essential role in efficiently handling these iterative and recurrent tasks.

For this work, we test the presented methodology for LC models deploying the MLOps environment on a Kubernetes cluster using open-source components. All the code needed for the retraining, registry, and deployment of the model in the environment is available in a GitHub repository,[5] which also includes all the CI/CD workflows and tests developed. Next, we will discuss the critical aspects of integrating annual LC models into the presented MLOps framework.

Every version of our annual LC models, from the initial baseline to each subsequent version, is meticulously documented in the model registry. Adhering to the principle of reproducibility, each model version is accompanied by comprehensive documentation, including training parameters, a confusion matrix, training and testing datasets, and the specific code used, along with its SHA reference in the GIT repository. Additionally, we include detailed records of Python packages with their respective versions and the model itself in binary format (refer to Fig. 3). A critical goal is always to have the best-performing version of the model deployed. Leveraging the model registry's comprehensive records and metadata, identifying and deploying the most effective model version can be efficiently automated. This ensures our system consistently operates with the most accurate and up-to-date model.

[4] https://doi.org/10.5281/zenodo.7462308
[5] https://github.com/KhaosResearch/mlops-landcoverpy

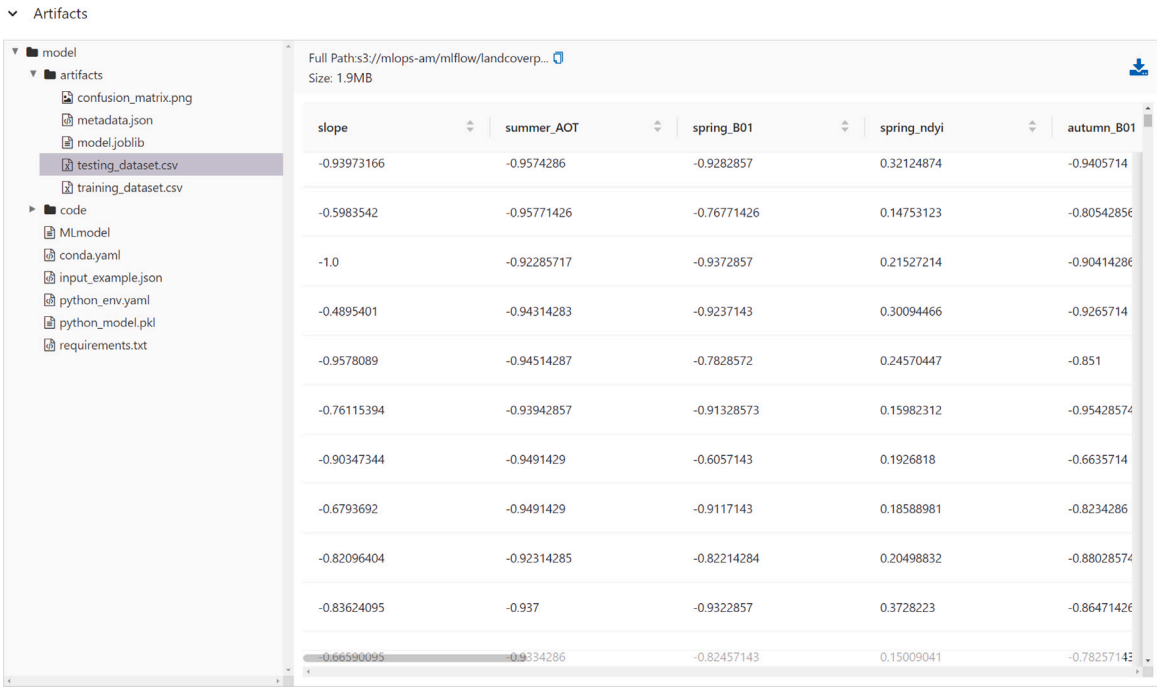


Fig. 3. GUI of MLflow [13] Model Registry showing stored artefacts of the LC model. Artefacts are usually heavy files (code, datasets) saved in object storage (e.g., S3). This model holds everything needed to replicate the training process. It includes a confusion matrix image, the model file, testing and training datasets, code used, and environment configuration files. Parameters and metrics do not appear in the picture but are stored. However, those are stored in a relational database (e.g., PostgreSQL [65]).

A model retraining pipeline has been established to process newly validated geolocated datasets. This pipeline accesses Sentinel-2 and ASTER product data from specified regions stored in S3, creating a tabular dataset with the 15 features defined in the baseline model. This new dataset is then merged with data previously used for training the current production model, retrievable from the model registry. A new model version is trained and subsequently registered in the model registry upon merging. Performance metrics of the new and production versions are compared, with the latest version being automatically redeployed if it surpasses the current performance. This process ensures that the workflow is automatically activated upon the availability of new data, maintaining the model's continuous update and improvement with each data integration.

Deployed model instances are accessible to users through gRPC or REST APIs, facilitating seamless interaction with annual models. Users can use these APIs to submit a geographical area defined by latitude and longitude coordinates in geoJSON format. This geometry is then transmitted to a model instance, which conducts an LC prediction for the designated area in the requested year. The prediction output is subsequently stored in an S3 bucket. Upon completion of the processing, users are provided with a public link to this object, enabling them to download a raster that visualises the classified zone with a 10 m spatial resolution. This raster reflects the nine distinct classes identified by the model, thus providing users with detailed and valuable LC classifications tailored to their specified areas. A prediction example is shown in Fig. 1(a).

Whenever a user prediction is sent to an instance of the served models, they export real-time metrics about the forecast to a designated port. Among the metrics designed for the workflow, notable ones are the average confidence of the model in the classified area, the number of predictions made by the model instance, or the response time of the REST/gRPC request.

The average confidence of the model is computed using the maximum probability from the prediction probabilities of each pixel in the area of interest across all classes. This is expressed mathematically as follows:

$$\text{AvgConf} = \frac{1}{N}\sum_{i=1}^{N} \max\left(P(\text{Class}_k|\text{Pixel}_i)\right) \tag{1}$$

where N is the number of pixels in the predicted area, $P(\text{Class}_k|\text{Pixel}_i)$ represents the probability of the most likely class k for the pixel at position (i).

Since validated data is typically unavailable in production environments, metrics such as accuracy or mean squared error are not suitable for these scenarios. Instead, the average confidence is useful as it does not require validated data to function, which is usually the case in real-time environments. If the model receives a prediction request with data significantly different from what it was trained on, this metric is likely to yield a low result.

These real-time metrics can be easily scraped using any monitoring system, such as Prometheus, and visualised using any metric visualisation system, like Grafana [68]. Both methods are open-source and supported in Kubernetes.

While all these services are interconnected, it is essential to establish rules for triggering the entire training process. The specific use case of the model should determine these conditions. In our use case, we have employed the average confidence of last-day predictions. If the daily average falls below a certain threshold, an alert is triggered to initiate new training if new data is available. This alert serves not only to retrain the model but also for experts to assess its performance and the results of possibly inaccurate predictions. Additionally, the model can be directly replaced in production or launched as a shadow deployment for expert evaluation of the behaviour of the newly trained model before replacing the production model. All of these decisions are contingent upon the specifics of the use case.

Our implementation includes GitHub Actions workflows[6] that automatically update the required containers used in the MLOps framework whenever there are changes in the code that affect them. All containers are automatically uploaded to the GitHub Container Registry (GHCR).

[6] https://docs.github.com/en/actions/using-workflows

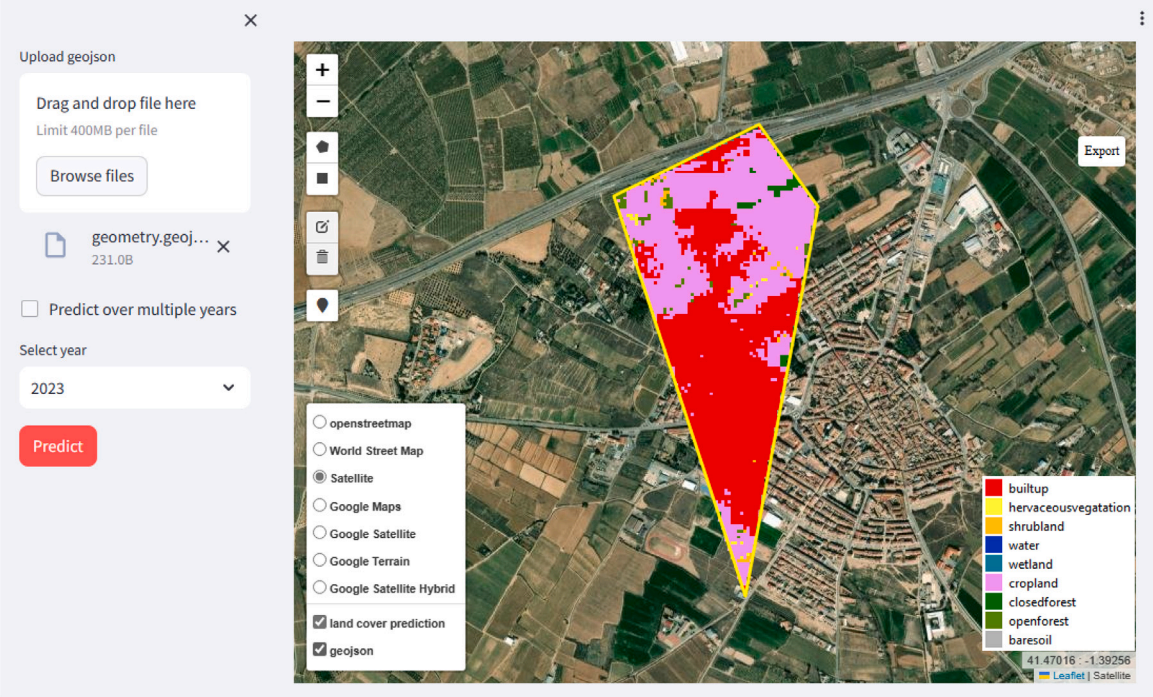


Fig. 4. GUI for LC Prediction. The interface allows users to upload a GeoJSON file delineating the AOI (left panel) and select the desired year for LC prediction. Upon clicking 'Predict', a gRPC request is sent to any deployed instance of the corresponding year. Then, the model processes the data and overlays the resulting LC classification on the map, colour-coded according to the legend (right panel). Each colour represents a different LC category, such as built-up areas, herbaceous vegetation, water, and various types of forestation, against the backdrop of a satellite image for geographical context. The GUI also allows the export of the prediction results and supports multiple base map layers for enhanced visualisation.

In this way, we meet the requirements of continuous integration. When a newly trained model arrives, redeploying the model in the Kubernetes cluster fulfils the need for constant deployment.

5. Results

To validate our methodology, we develop a prototype user interface that simplifies end-user interaction with the API. A performance test is also conducted to simulate real-world distributed usage while monitoring the entire system using predefined metrics. This test is crucial for verifying the monitoring system, deployed instances' performance, and real-time metrics' accuracy. Furthermore, manual trigger tests with newly validated data were executed. These tests validated the entire system, confirming the efficiency of the retraining workflow, the model registry process, and the deployment of the new version, thereby ensuring the seamless functioning of the complete system.

The Kubernetes cluster used in this experiment consists of three virtual machines created on a server within our local infrastructure. Each virtual machine, including one master node, without a scheduling taint, and two worker nodes, is equipped with 8 virtual CPU cores and approximately 64 GB of RAM. These virtual CPU cores are provisioned from a physical Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60 GHz processor. All nodes run on Ubuntu 22.04 LTS and use containerd as the container runtime. The Kubernetes version on all nodes is v1.27.2.

The web-based user interface is developed using the Python library Streamlit [69], enabling end-users to delineate an AOI on an interactive map and perform LC predictions for a specific year or multiple years simultaneously. Once a prediction is completed, the results are loaded onto the map as new layers, following a predefined colour legend. Users can download any of these predictive layers in the .tif format, a raster format used by several Geographic Information Systems (GIS). Fig. 4 illustrates an example of such a prediction made using this user interface.

To rigorously evaluate the efficacy and performance of our system, a comprehensive test is conducted involving the distribution of 1000 prediction requests across a ten replica model deployed within our cluster. Grafana, a powerful analytic and monitoring solution, visualises these metrics in real-time, providing an insightful and detailed overview of the system's performance under simulated operational conditions. The results are depicted in Fig. 5, offering a granular analysis of the system's robustness. The data includes the fluctuating confidence levels that each pod experienced throughout the testing period, which reflects the model's predictive consistency and reliability.

The cumulative count of predictions by pod illustrates the system's load-balancing efficacy and the capacity of each replica to handle incoming requests. This testing validates the system's stability and scalability and helps identify potential areas for optimisation. The insights gained from this experiment are instrumental in guiding further refinements to ensure that the system performs optimally under varied and intensive workloads.

The previous results already indicate the proper functioning of the entire workflow. However, further verification was conducted by manually triggering the process. This step involved a comprehensive review to confirm the successful execution of the retraining pipeline, resulting in a new model version uploaded to the model registry and subsequently deployed in the Kubernetes cluster as a shadow deployment.

6. Discussion

In this work, we face numerous challenges, each demanding a unique strategy for resolution. While some challenges lacked perfect solutions, we explored various alternatives, each with advantages and disadvantages.

The primary source of these challenges is the use of high-resolution satellite images. Each Sentinel-2 10 m band can be as large as 150MB, and vegetation indices require multiple bands' combination (and thus

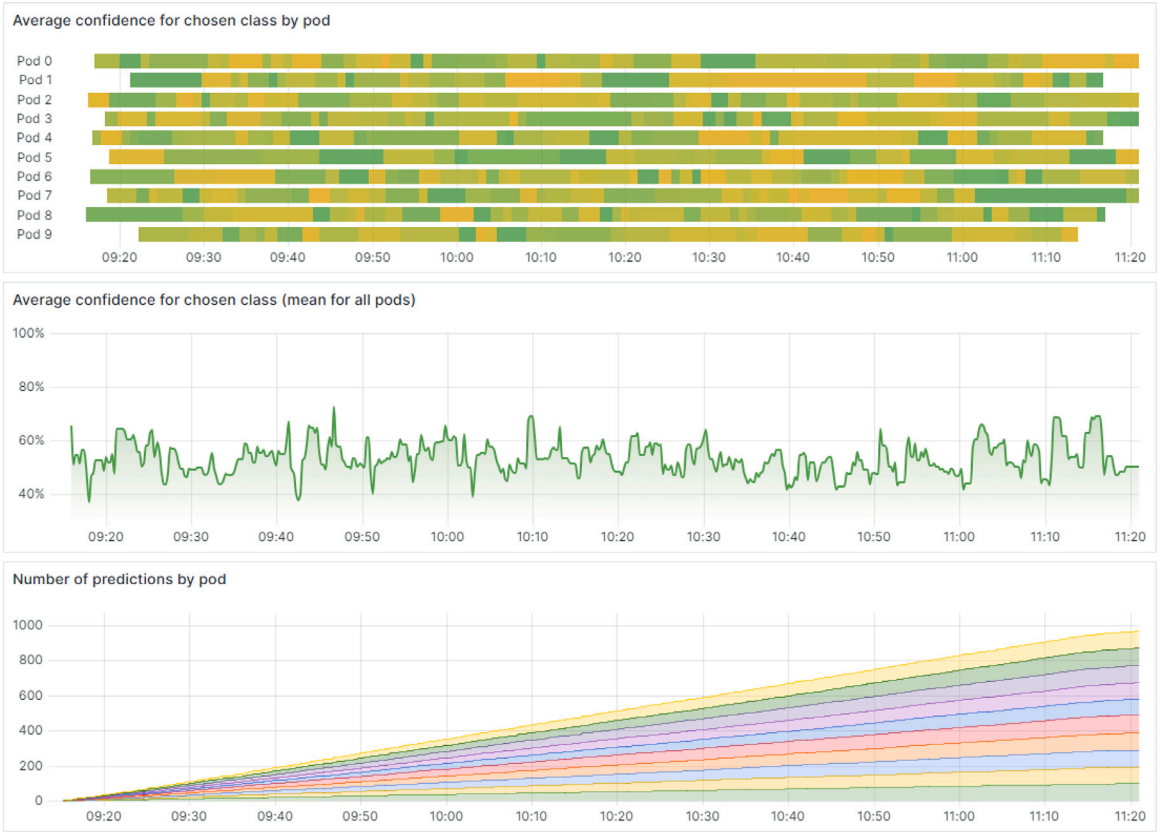


Fig. 5. Metrics obtained from 1000 user prediction requests directed to 10 deployed replicas of the model. The top chart displays the average confidence level for predicting each pixel, categorised by pod, with green indicating high confidence (70% or higher) and red indicating low confidence (30% or lower). The middle chart showcases the average confidence level across all pods. The bottom chart illustrates the cumulative number of predictions, ranging from 0 to 1000. The entire test was conducted over 2 h. *Note that predictions stop at ≈970. Some predictions failed due to a lack of Sentinel-2 products in those areas.* The 1000 geoJSONs used were automatically generated to cover various shapes, areas, and locations.

reading). The significant issue here is not the memory needed for the microservice instance, as optimisation techniques can reduce this, like setting a maximum area for prediction per model instance and distributing it among instances. Instead, the bottleneck is the time to read these large data volumes. If a model must read 1 GB of data for each prediction, it results in a significant bottleneck that depends on multiple factors.

Our initial model had 78 features, including Sentinel-2 bands from different seasons, spectral indices, and terrain data from the ASTER satellite. Deploying such a feature-intensive model within an MLOps framework proved impractical due to the prolonged prediction times it would impose on end users. To optimise performance without overly compromising accuracy, we employ Recursive Feature Elimination, streamlining the model by discarding redundant features and enhancing efficiency while carefully managing the trade-off with classification precision.

The official method for reading images is through the Sentinel API [70]. However, this approach requires downloading a compressed file for each product containing all bands, involving downloading unnecessary data. We consider two alternatives: They are first, using private initiatives like Google Cloud Storage,[7] which mirrors the data available on the API. This method enables access to individual, uncompressed bands and offers worldwide coverage. Its drawbacks include dependency on external servers and the potential variability of data availability and download speeds. Secondly, we consider automating band downloads directly to Kubernetes cluster-mounted storage, such as S3. This significantly reduces download times but requires substantial data storage capacity. At a 10 m resolution, the data storage demands could reach several petabytes, leading to high costs, mainly when accounting for data replication and distribution. We recommend this latter approach for projects with a limited geographical scope or temporal range, such as country-level studies or analyses confined to a specific timeframe.

The size of Sentinel-2 tiles further compounds the reading time problem, each covering 110 × 110 km^2 (each 10 m band containing approximately 121 million pixels). A strategic optimisation, if data is downloaded and stored, is to subdivide the large Sentinel-2 tiles into smaller segments. This approach aligns with the typical scale of prediction requests, which often target smaller land areas. This shortens read times, facilitates requests across instances, and decreases the model's minimum memory space requirements.

Another major issue in remote sensing is cloud cover, typically addressed using compositing several bands of different dates. Generating these composites on the fly during prediction requests can significantly extend processing times. A practical solution is to precompute and store these composites in advance rather than the individual bands. This precomputation can be synchronised with the subdivision of bands, further streamlining the prediction workflow and reducing latency. Such preparedness allows immediate access to cloud-free imagery, expediting the prediction process.

In summary, most problems can be solved by creating internal storage preprocessing Sentinel-2 products, adapting them to the use case or optimising the model. The more the base satellite images are processed, the more efficient the system becomes. However, it is challenging to do this globally at a 10 m scale due to physical or cloud storage costs. To overcome this issue, a hybrid strategy could be

[7] https://cloud.google.com/storage/docs/public-datasets/sentinel-2

employed. This would involve optimising data for specifically targeted areas and periods of interest while relying on alternative solutions such as Google Cloud Storage for broader coverage. Such an approach offers a balanced solution, combining the benefits of global coverage with the efficiency of localised data optimisation.

7. Conclusions

In this article, an MLOps framework using open-source software has been developed, adhering to established standards to explore the integration of ML models dealing with satellite imagery. Specifically, we focus on models that process high spatial resolution data, addressing the significant challenge of managing Big Data in this context. A novel methodology has been successfully incorporated into this MLOps framework for handling annual LC data, and its functionality has been validated through various tests. Our results reveal that an end-to-end system for predicting annual LC with high spatial resolution is achievable by applying MLOps practices within a containerised environment, offering on-demand auto-scaling, efficient model management, and distributed deployment.

In advancing the MLOps framework, future research endeavours will concentrate on two pivotal fronts: incorporating a robust drift detection mechanism and integrating an explicability component. Regarding drift detection, we aim to implement state-of-the-art techniques that can autonomously identify and alert about shifts in data distributions over time, thereby ensuring the sustained accuracy and reliability of ML models. This proactive approach will bolster the framework's adaptability in environments where data shifts are expected. Simultaneously, the growing importance of model explainability is widely recognised. Thus, the following step incorporates interpretable AI components, enabling the comprehension and trust of ML models' decision-making processes. By combining drift detection and explainability, we anticipate further advancing the efficacy and transparency of the MLOps framework, contributing to the ongoing progress in MLOps research.

CRediT authorship contribution statement

Antonio M. Burgueño-Romero: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Cristóbal Barba-González:** Writing – review & editing, Writing – original draft, Visualization, Methodology, Conceptualization. **José F. Aldana-Montes:** Writing – review & editing, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that support the findings of this study are openly available in GitHub repositories https://github.com/KhaosResearch/landcoverpy , https://github.com/KhaosResearch/mlops-landcoverpy and https://github.com/KhaosResearch/mlops-infra.

Declaration of Generative AI and AI/assisted technologies in the writing process

Statement: During the preparation of this work the authors used GPT-3.5 in order to enhance the clarity and correctness of the language used in the manuscript. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgements

This work has been funded by grant PID2020-112540RB-C41,[8] 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.

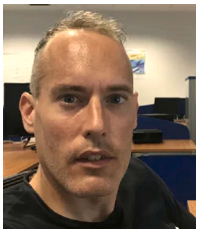
References

[1] G. Cheng, X. Xie, J. Han, L. Guo, G.-S. Xia, Remote sensing image scene classification meets deep learning: Challenges, methods, benchmarks, and opportunities, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 13 (2020) 3735–3756.
[2] T.N. Phan, V. Kuch, L.W. Lehnert, Land cover classification using google earth engine and random forest classifier—The role of image composition, Remote Sens. 12 (15) (2020) 2411.
[3] A. Mercier, J. Betbeder, F. Rumiano, J. Baudry, V. Gond, L. Blanc, C. Bourgoin, G. Cornu, C. Ciudad, M. Marchamalo, et al., Evaluation of sentinel-1 and 2 time series for land cover classification of forest–agriculture mosaics in temperate and tropical landscapes, Remote Sens 11 (8) (2019) 979.
[4] Y. Zhang, L. Sun, Spatial-temporal impacts of urban land use land cover on land surface temperature: Case studies of two Canadian urban areas, Int. J. Appl. Earth Obs. Geoinformation 75 (2019) 171–181.
[5] H. Aghsaei, N.M. Dinan, A. Moridi, Z. Asadolahi, M. Delavar, N. Fohrer, P.D. Wagner, Effects of dynamic land use/land cover change on water resources and sediment yield in the Anzali wetland catchment, Gilan, Iran, Sci. Total Environ. 712 (2020) 136449.
[6] S. Sun, Z. Cao, H. Zhu, J. Zhao, A survey of optimization methods from a machine learning perspective, IEEE Trans. Cybern. 50 (8) (2019) 3668–3681.
[7] Z.-H. Zhou, Machine learning, Springer Nature, 2021.
[8] I.H. Sarker, Machine learning: Algorithms, real-world applications and research directions, SN Comput. Sci. 2 (3) (2021) 160.
[9] L. Zhou, S. Pan, J. Wang, A.V. Vasilakos, Machine learning on big data: Opportunities and challenges, Neurocomputing 237 (2017) 350–361.
[10] X.-S. Wei, Y.-Z. Song, O. Mac Aodha, J. Wu, Y. Peng, J. Tang, J. Yang, S. Belongie, Fine-grained image analysis with deep learning: A survey, IEEE Trans. Pattern Anal. Mach. Intell. 44 (12) (2021) 8927–8948.
[11] M.M. Najafabadi, F. Villanustre, T.M. Khoshgoftaar, N. Seliya, R. Wald, E. Muharemagic, Deep learning applications and challenges in big data analytics, J. Big Data 2 (1) (2015) 1–21.
[12] L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316.
[13] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S.A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al., Accelerating the machine learning lifecycle with mlflow., IEEE Data Eng. Bull. 41 (4) (2018) 39–45.
[14] C. Ebert, G. Gallardo, J. Hernantes, N. Serrano, DevOps, Ieee Softw. 33 (3) (2016) 94–100.
[15] L. Bass, I. Weber, L. Zhu, DevOps: A software architect's perspective, Addison-Wesley Professional, 2015.
[16] J. Pineda-Jaramillo, F. Viti, Mlops in freight rail operations, Eng. Appl. Artif. Intell. 123 (2023) 106222.
[17] L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles, A survey of DevOps concepts and challenges, ACM Comput. Surv. 52 (6) (2019) 1–35.
[18] R. Subramanya, S. Sierla, V. Vyatkin, From DevOps to MLOps: Overview and application to electricity market forecasting, Appl. Sci. 12 (19) (2022) 9851.
[19] A. Masood, A. Hashmi, A. Masood, A. Hashmi, AIOps: predictive analytics & machine learning in operations, Cogn. Comput. Recipes Artif. Intell. Solut. Microsoft Cogn. Serv. TensorFlow (2019) 359–382.
[20] N. Memon, H. Parikh, S.B. Patel, D. Patel, V.D. Patel, Automatic land cover classification of multi-resolution dualpol data using convolutional neural network (CNN), Remote Sens. Appl. Soc. Environ. 22 (2021) 100491.
[21] H.R. Goldberg, C.R. Ratto, A. Banerjee, M.T. Kelbaugh, M. Giglio, E.F. Vermote, Automated global-scale detection and characterization of anthropogenic activity using multi-source satellite-based remote sensing imagery, in: Geospatial Informatics XIII, 12525, SPIE, 2023, 1252502.
[22] M. Luksa, Kubernetes in Action, Simon and Schuster, 2017.
[23] L. Toka, G. Dobreff, B. Fodor, B. Sonkoly, Machine learning-based scaling management for kubernetes edge clusters, IEEE Trans. Netw. Serv. Manag. 18 (1) (2021) 958–972.
[24] S. Roh, K.-M. Jeong, H.-Y. Cho, E.-N. Huh, An efficient microservices architecture for mlops, in: 2023 Fourteenth International Conference on Ubiquitous and Future Networks, ICUFN, IEEE, 2023, pp. 652–654.
[25] K. Salama, J. Kazmierczak, D. Schut, Practitioners guide to mlops: A framework for continuous delivery and automation of machine learning, 2021, Google Could White paper.

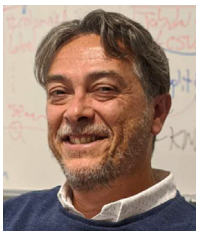
[8] Funded by MCIN/AEI/10.13039/501100011033/.

[26] A.M. Burgueño, J.F. Aldana-Martín, M. Vázquez-Pendón, C. Barba-González, Y. Jiménez Gómez, V. García Millán, I. Navas-Delgado, Scalable approach for high-resolution land cover: a case study in the Mediterranean basin, J. Big Data 10 (1) (2023) 1–22.
[27] Y. Dang, Q. Lin, P. Huang, Aiops: real-world challenges and research innovations, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2019, pp. 4–5.
[28] B. Fitzgerald, K.-J. Stol, Continuous software engineering: A roadmap and agenda, J. Syst. Softw. 123 (2017) 176–189.
[29] P. Notaro, J. Cardoso, M. Gerndt, A survey of AIOps methods for failure management, ACM Trans. Intell. Syst. Technol. 12 (6) (2021) 1–45.
[30] L. Yang, D. Rossi, Quality monitoring and assessment of deployed deep learning models for network AIOps, IEEE Netw. 35 (6) (2021) 84–90.
[31] H.B.r. Christensen, Teaching DevOps and cloud computing using a cognitive apprenticeship and story-telling approach, in: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, 2016, pp. 174–179.
[32] D. Kreuzberger, N. Kühl, S. Hirschl, Machine learning operations (mlops): Overview, definition, and architecture, IEEE Access (2023).
[33] A. Mishra, Z. Otaiwi, DevOps and software quality: A systematic mapping, Comp. Sci. Rev. 38 (2020) 100308.
[34] G. Symeonidis, E. Nerantzis, A. Kazakis, G.A. Papakostas, Mlops-definitions, tools and challenges, in: 2022 IEEE 12th Annual Computing and Communication Workshop and Conference, CCWC, IEEE, 2022, pp. 0453–0460.
[35] M.M. John, H.H. Olsson, J. Bosch, Towards mlops: A framework and maturity model, in: 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, IEEE, 2021, pp. 1–8.
[36] B.M. Matsui, D.H. Goya, Mlops: five steps to guide its effective implementation, in: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI, 2022, pp. 33–34.
[37] S. Mäkinen, H. Skogström, E. Laaksonen, T. Mikkonen, 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, IEEE, 2021, pp. 109–112.
[38] C. Renggli, L. Rimanic, N.M. Gürel, B. Karlaš, W. Wu, C. Zhang, A data quality-driven view of mlops, 2021, arXiv preprint arXiv:2102.07750.
[39] P. Ruf, M. Madan, C. Reich, D. Ould-Abdeslam, Demystifying mlops and presenting a recipe for the selection of open-source tools, Appl. Sci. 11 (19) (2021) 8861.
[40] J. Klaise, A. Van Looveren, C. Cox, G. Vacanti, A. Coca, Monitoring and explainability of models in production, 2020, arXiv preprint arXiv:2007.06299.
[41] D.A. Tamburri, Sustainable mlops: Trends and challenges, in: 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, IEEE, 2020, pp. 17–23.
[42] I. Pölöskei, Mlops approach in the cloud-native data pipeline design, Acta Technica Jaurinensis 15 (1) (2022) 1–6.
[43] M. Reddy, B. Dattaprakash, S. Kammath, S. Kn, S. Manokaran, R. Be, Application of mlops in prediction of lifestyle diseases, ECS Trans. 107 (1) (2022) 1191.
[44] G. Büttner, CORINE land cover and land cover change products, Land Use Land Cover. Mapp. Eur. Pract. Trends (2014) 55–74.
[45] S.P. Abercrombie, M.A. Friedl, Improving the consistency of multitemporal land cover maps using a hidden Markov model, IEEE Trans. Geosci. Remote Sens. 54 (2) (2015) 703–713.
[46] A. Mousivand, J.J. Arsanjani, Insights on the historical and emerging global land cover changes: The case of ESA-CCI-LC datasets, Appl. Geogr. 106 (2019) 82–92.
[47] K. Karra, C. Kontgis, Z. Statman-Weil, J.C. Mazzariello, M. Mathis, S.P. Brumby, Global land use / land cover with sentinel 2 and deep learning, in: 2021 IEEE International Geoscience and Remote Sensing Symposium, IGARSS, 2021, pp. 4704–4707, http://dx.doi.org/10.1109/IGARSS47720.2021.9553499.
[48] Z.S. Venter, D.N. Barton, T. Chakraborty, T. Simensen, G. Singh, Global 10 m land use land cover datasets: A comparison of dynamic world, world cover and esri land cover, Remote Sens. (ISSN: 2072-4292) 14 (16) (2022) http://dx.doi.org/10.3390/rs14164101, URL https://www.mdpi.com/2072-4292/14/16/4101.
[49] M.G. Tulbure, P. Hostert, T. Kuemmerle, M. Broich, Regional matters: On the usefulness of regional land-cover datasets in times of global change, Remote Sens. Ecol. Conserv. 8 (3) (2022) 272–283, http://dx.doi.org/10.1002/rse2.248, arXiv:https://zslpublications.onlinelibrary.wiley.com/doi/pdf/10.1002/rse2.248, URL https://zslpublications.onlinelibrary.wiley.com/doi/abs/10.1002/rse2.248.
[50] L. Tian, R. Sedona, A. Mozaffari, E. Kreshpa, C. Paris, M. Riedel, M.G. Schultz, G. Cavallaro, End-to-end process orchestration of earth observation data workflows with apache airflow on high performance computing, in: IGARSS 2023-2023 IEEE International Geoscience and Remote Sensing Symposium, IEEE, 2023, pp. 711–714.
[51] G. Giacco, S. Marrone, G. Langella, C. Sansone, Refuse: Generating imperviousness maps from multi-spectral sentinel-2 satellite imagery, Future Internet 14 (10) (2022) 278.
[52] Z. Sun, N. Cristea, Introduction of artificial intelligence in earth sciences, in: Artificial Intelligence in Earth Science, Elsevier, 2023, pp. 1–15.
[53] M. Luotamo, et al., Advances in region-based multisource machine learning for remote sensing, Series Publ. (2023).
[54] MLflow, MLflow: An open source machine learning platform, 2023, URL https://mlflow.org/ Accessed 18 September 2023.
[55] Neptune.ai, Neptune.ai: The machine learning platform, 2023, URL https://neptune.ai/ Accessed 18 September 2023.
[56] Y. Zhou, Y. Yu, B. Ding, Towards MLOps: A case study of ML pipeline platform, in: 2020 International Conference on Artificial Intelligence and Computer Engineering, ICAICE, 2020, pp. 494–500, http://dx.doi.org/10.1109/ICAICE51518.2020.00102.
[57] Prefect, Prefect: The dataflow automation platform, 2023, URL https://www.prefect.io/ Accessed 18 September 2023.
[58] Luigi Documentation, Python module that helps you build complex pipelines of batch jobs, 2023, URL https://luigi.readthedocs.io/ Accessed 18 September 2023.
[59] Apache Airflow, Apache airflow: A platform to programmatically author, schedule, and monitor workflows, 2023, URL https://airflow.apache.org/ Accessed 18 September 2023.
[60] Seldon Technologies, Seldon: Deploy, monitor, and scale machine learning models, 2023, URL https://www.seldon.io/ Accessed 18 September 2023.
[61] BentoML, BentoML: A framework for machine learning model serving, 2023, URL https://www.bentoml.com/ Accessed 18 September 2023.
[62] TensorFlow, TensorFlow: An open source machine learning framework for everyone, 2023, URL https://www.tensorflow.org/ Accessed 18 September 2023.
[63] Prometheus, Prometheus: An open-source monitoring and alerting toolkit, 2023, URL https://prometheus.io/ Accessed 18 September 2023.
[64] Zabbix, Zabbix: The enterprise-class monitoring solution for everyone, 2023, URL https://www.zabbix.com/ Accessed 18 September 2023.
[65] B. Momjian, PostgreSQL: introduction and concepts, vol. 192, Addison-Wesley New York, 2001.
[66] M. Immitzer, F. Vuolo, C. Atzberger, First experience with sentinel-2 data for crop and tree species classifications in central europe, Remote Sens 8 (3) (2016) 166.
[67] M. Abrams, R. Crippen, H. Fujisada, ASTER global digital elevation model (GDEM) and ASTER global water body dataset (ASTWBD), Remote Sens. (ISSN: 2072-4292) 12 (7) (2020) http://dx.doi.org/10.3390/rs12071156, URL https://www.mdpi.com/2072-4292/12/7/1156.
[68] Grafana, Grafana: The open observability platform, 2024, URL https://grafana.com/ Accessed 8 January 2024.
[69] Streamlit, Streamlit: A faster way to build and share data apps, 2024, URL https://streamlit.io Accessed 8 January 2024.
[70] Copernicus DataSpace, Copernicus sentinel-2 data documentation, 2023, https://documentation.dataspace.copernicus.eu/Data/Sentinel2.html Accessed 20 December 2023.

Antonio M. Burgueño-Romero is currently pursuing a Ph.D. in the Department of Languages and Computer Sciences at the University of Málaga, conducting his research as part of Khaos Research Group. Burgueño has papers published in a wide variety of fields, such as: Remote Sensing, Computer Vision, Robotics or Artificial Intelligence. Right now, he is focused in the areas of MLOps and distributed applications.



Cristóbal Barba-González is a Ph.D. and assistant professor at the University of Málaga, Spain. His areas of interest are Big Data analysis; he mainly focuses on Machine Learning algorithms and streaming processes, parallel computing, multi-objective optimisation with metaheuristics, and domain knowledge within Big Data analytics algorithms.



José F. Aldana-Montes is a full professor of the Computer Languages and Systems area in the Department of Languages and Computer Sciences of the University of Málaga. With more than 30 years of experience as a teacher in the area of databases and related areas. His areas of interest are Semantic Middleware, Semantic Web, Semantic Integration of Data and Applications, Extensions of the Databases with Formal Semantics and Big Data analysis.