DESIGN AND DEVELOPMENT OF AN MLOPS FRAMEWORK

IAGO ÁGUILA CIFUENTES

Thesis supervisor: JAVIER BÉJAR ALONSO (Department of Computer Science)

Thesis co-supervisor: TANIA KLYMCHUK (SENIOR AI DEVELOPER AT BASETIS)

Degree: Master Degree in Artificial Intelligence

Thesis report

School of Engineering
Universitat Rovira i Virgili (URV)

Faculty of Mathematics
Universitat de Barcelona (UB)

Barcelona School of informatics (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Abstract

This thesis explores the role of MLOps in providing efficiency and productivity in the deployment, monitoring and management of machine learning models in production environments. The report has a theoretical part that aims to provide details on the operation of MLOps practices and related technologies. This serves as a complement for the practical part, which is considered the main contribution. It consists of the development of a framework that aims to collect some of the main functionalities of MLOps. This helps to meet the main objective of demonstrate its usefulness and to understand why MLOps is increasingly important. The development of the framework involves the use of Python, Docker, Streamlit and Airflow, each one necessary to provide different MLOps features.

Keywords: MLOps; DevOps; machine learning; framework; continuous training; continuous monitoring; production environment.

Resum

Aquesta tesi explora el paper de MLOps per a proporcionar eficiència i productivitat en el desplegament, monitoratge i gestió de models d'aprenentatge automàtic en entorns de producció. L'informe compta amb una part teòrica que pretén aportar detalls sobre el funcionament de les pràctiques de MLOps i les tecnologies relacionades. Això serveix de complement per a la part pràctica, la qual es considera la principal contribució. Consisteix en el desenvolupament d'un framework que pretén recollir algunes de les principals funcionalitats de MLOps. Això ajuda a complir l'objectiu principal de demostrar la seva utilitat i comprendre per què MLOps és cada vegada més important. El desenvolupament del framework implica l'ús de Python, Docker, Streamlit i Airflow, cadascun necessari per a proporcionar diferents funcionalitats de MLOps.

Paraules clau: MLOps; DevOps; aprenentatge automàtic; framework; entrenament continu; monitoratge continu; entorn de producció.

Resumen

Esta tesis explora el papel de MLOps para proporcionar eficiencia y productividad en el despliegue, monitorización y gestión de modelos de aprendizaje automático en entornos de producción. El informe cuenta con una parte teórica que pretende aportar detalles sobre el funcionamiento de las prácticas de MLOps y las tecnologías relacionadas. Esto sirve de complemento para la parte práctica, la cuál se considera la principal contribución. Consiste en el desarrollo de un framework que pretende recoger algunas de las principales funcionalidades de MLOps. Esto ayuda a cumplir el objetivo principal de demostrar su utilidad y comprender por qué MLOps es cada vez más importante. El desarrollo del framework implica el uso de Python, Docker, Streamlit y Airflow, cada uno necesario para proporcionar diferentes funcionalidades de MLOps.

Palabras clave: MLOps; DevOps; aprendizaje automático; framework; entrenamiento continuo; monitorización continua; entorno de producción.

Acknowledgments

To my family and friends, thank you for the unconditional support you have given me throughout my academic life, your sense of pride keeps me going.

To my former coworkers, thank you for giving me the confidence and support when I made my decision to take a leap of faith and do the master's degree. It was worth it. To the professors and students with whom I have coincided, thanks for your dedication and companionship. This stage is more bearable and enjoyable thanks to you. To my tutor Javier Béjar Alonso, for accepting the proposal of this thesis and showing me his support.

Finally, thanks to Pau Coll, Tetiana Klymchuk, Marc Alvinyà and Àlex Ferrer for introducing me to a very interesting area of artificial intelligence, MLOps. Besides generating me new opportunities, thanks to you I have had the occasion to present this thesis.

Thanks to all of you,

Iago.

Contents

List of Figures

1 Introduction

Machine learning is a powerful tool capable of solving problems in a large number of different fields, such as healthcare, finance, marketing, robotics, etc. Its ability to accurately predict outcomes from historical data is what makes it such a widely used tool. Moreover, machine learning is becoming more and more important day by day in a society that is becoming increasingly data-driven. Many of today's leading companies, such as Facebook, Google and Uber, make machine learning a central part of their operations, which is an indicator that machine learning has become a significant competitive differentiator for many companies [8].

Data preparation, model training and model evaluation are some of the tasks involved in the complex process of creating machine learning models in a development environment. The goal of a machine learning project is for the generated model to work in a production (real world) environment, which makes the deployment of the model also part of the process. It is common to find situations where the model works correctly during development and testing, but when deployed to a production environment the desired behavior is not achieved. This can be caused by different reasons, such as data drift, where the data that the model receives in production does not accurately reflect the data it has been trained with. The management and deployment of models in production environments must be done in an efficient manner in order to help solve these issues. This is where MLOps comes in.

MLOps, or Machine Learning Operations, is an extension of the DevOps methodology, which is a continuous software engineering practice [15]. DevOps, or Development Operations, is a set of practices that brings together development and IT operations. A key aspect is that it provides continuous delivery and continuous deployment so that the software is delivered faster with short delivery cycles [4]. DevOps frameworks also ensure reliability and repeatability through automation. With code-centric applications, where it is simple to test whether they work as expected, DevOps is used since it is able to perform tasks like ensuring data integrity and validity. The data-centric nature of machine learning and artificial intelligence models, in contrast, adds new problems that conventional DevOps cannot address [27]. In these situations, MLOps is applied, which imitates DevOps procedures while adding extra steps that are unique to the machine learning field.

MLOps is a set of practices that aims to maintain and deploy machine learning code and models with high reliability and efficiency. In a general way, it is able to manage the ML lifecycle, which refers to the complete process of developing, deploying and maintaining the ML application. The main objective of MLOps is to achieve faster development and deployment of machine learning models with high quality, reproducibility and end-to-end tracking [34]. Automated testing and retraining are examples of actions provided by MLOps, which in this case help monitor and update models continuously so that they can evolve with the data. Overall, implementing an MLOps framework can provide a number of benefits such as increased productivity, reliability, data and model quality, among others.

The origin of DevOps dates back to 2007 and 2008, when the IT operations and software development model communities objected to the fact that the teams in charge of writing code and those in charge of deploying and supporting that code were organizationally and functionally separated [7]. On the other hand, MLOps is a more recent set of practices, taking into account that it is considered to be the "natural progression of

DevOps in the context of AI", which was mentioned by Samir Tout who is a professor of cybersecurity at the Eastern Michigan University's School of Information Security and Applied Computing (SISAC) [35]. The MLOps movement began in 2015 with the publication of a paper called "Hidden Technical Debt in Machine Learning", which discusses and analyzes the costs of maintaining real-world machine learning systems [31]. Since then, MLOps has grown considerably in the market, although its largest growth is predicted to occur in the next few years. Furthermore, the current trend of machine learning and artificial intelligence accompanies and contributes to this growth [32].

The paper entitled "Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?" [23], mentions a type of organization that has diverse models in production and is increasingly business-critical. The paper categorizes these organizations as pipeline-centric, which constantly consider how to scale and do continuous development while at the same time maintaining the quality of production models. The authors mention that these organizations are the ones where MLOps can bring the most value at the level of daily operations. However, interest in MLOps exists in other types of organizations that are, for example, focused on developing their first model and deploying it in production.

The increasing number of platforms and technologies that are available to enable MLOps illustrates how important it is. They include both commercial platforms like AWS SageMaker and Microsoft Azure Machine Learning as well as open-source solutions like Kubeflow, MLflow, and Metaflow. Model serving, pipeline orchestration, automated testing, and model versioning are just a few of the features and functionalities that these solutions offer to assist MLOps (host machine learning models on the cloud or on premises).

1.1 Objectives & Methodology

The main purpose of this thesis is to design and develop an MLOps framework that addresses the challenges of managing and maintaining machine learning models in a pre-production environment. The framework will integrate various practices such as data engineering, monitoring, evaluation and model training to ensure continuous improvement. The framework development is intended to demonstrate the effectiveness of MLOps practices. Is composed of three main components, being the first one a data generation and preparation module, which generates new data to simulate a real-world scenario. The second one is an orchestration module, which coordinates and automates tasks and operations related to monitoring, evaluating and training models. The last one is an user-facing dashboard, which provides visibility and control over the pipeline for the user.

The developed framework handles models that are trained with a particular dataset, called M5 Forecasting. The dataset is static, but is modified to add simulated instances over time. In this way, the performance of the models can be observed as the dataset generates new instances. In these cases, the monitoring provided by the MLOps framework is essential. Since the dataset is continually updated over time, it requires the implementation of preprocessing pipeline together with its periodical trigger. This implies a data engineering procedure to perform the dataset preprocessing. It is worth mentioning that it is not the main objective to perform a good preprocessing to properly train the models, since the thesis is focused on the performance of the MLOps framework. Anyway, the data engineering exercise is developed in a justified way according to the dataset and the task to be solved.

The user dashboard is an essential component of the proposed MLOps framework as it provides a user-friendly interface for monitoring, evaluating and controlling the pipeline. From the dashboard, the user is able to train the available models with new hyperparameters, evaluate the performance of different models, and monitoring them. The monitoring component helps the user to determine the most suitable model to run in production at any given time. The dashboard will also facilitate the visualization of the performance metrics of the models, making it easy for the user to identify any issues or opportunities for improvement. Additionally, the user will be able to schedule and execute tasks such as evaluation, training, and retraining, either automatically or manually. This possibility avoids a closed automated pipeline.

To implement this framework, a set of tools that includes Python, Airflow, Docker and Streamlit are used. Python is used for building the MLOps pipeline and the preprocessing pipeline, Airflow for scheduling and orchestration of the MLOps pipeline, Docker for containerization, and Streamlit for building the user dashboard. No cloud providers that offer MLOps solutions such as AWS SageMaker or Azure MLOps are used in the development. The objective is to develop a framework that works locally with the set of tools mentioned above. The introduction and theoretical explanation about MLOps are the complement to understand the context of the thesis and give importance to the development of the framework.

In order to develop the MLOps framework, a specific methodology has been followed. The first step consists of understanding the usefulness and functionality of MLOps, by consulting different sources of information. From here, an example case is defined in which MLOps is applicable. For this thesis, a series of trained models are working to solve the same task on a dataset that is updated over time, which is specifically a time series. In this situation, the monitoring and maintenance offered by an MLOps framework is a considerable advantage. Taking into account the casuistry that it is work with, the desired functionalities of MLOps are defined. For instance, one of the features to be implemented is the monitoring of the performance of each model, accompanied by automatic testing that is executed in an orchestrated manner.

After defining the scope and objectives of the framework, the necessary tools with which the implementation is carried out are selected. As previously mentioned, these are Python, Docker, Streamlit and Airflow. Before starting with the development of the code, different sources of information are consulted and the use of Docker, Streamlit and Airflow is put into practice, since they are the tools with which I personally had no previous experience.

Following the previous steps, which consist of organizing and planning the development of the MLOps framework and learning about new tools, the next step consists of developing the Python code. Starting with the most basic and essential, an analysis of the dataset is performed in order to build a preprocessing pipeline. From this task, a series of simple models are trained and everything is checked to ensure that it works correctly. By means of this basic functionality, the features provided by the tools used are added. For example, with the use of Streamlit a dashboard is built for the user and with the implementation of Airflow the orchestration of different tasks is achieved.

1.2 Motivation

During my curricular internship, I learned about MLOps together with the AI team, as it was a tool that all the team members had not yet used. It was an enriching experience for all participants and proved to be a beneficial tool, with the potential to propose a greater number of projects to possible clients. It is worth mentioning that part of the thesis methodology explained in the previous section was carried out in the internship, specifically the knowledge acquired about MLOps alongside the rest of the tools and the analysis and preprocessing pipeline of the dataset. This last task is not a main objective of the framework development, but it is used as a basis to show how it works.

From the knowledge acquired about MLOps, I have considered that it is a good opportunity to expand them in my master thesis, with the aim of developing a work that demonstrates to the community its usefulness. I think it is a favourable moment for the development of this work, as it is beneficial to expand my knowledge and skill in a field that is being recognized nowadays and, at the same time, to bring more recognition of it to the community. Also, for this same reason I believe that the timing for this thesis is appropriate.

In my personal experience, during my course in the master's degree in artificial intelligence, I have learned in a theoretical and practical way about different types of models and algorithms used in different fields such as natural language processing, deep learning, etc. In all cases, we have worked in a development environment as it is the best option for educational purposes. Doing a thesis related to the production environment is a good complement to completing the master's degree, as I become aware of the importance its importance.

The fact of giving more recognition to these practices is a motivation to develop the thesis, since initially data science was approached from an experimental and scientific perspective. Today, due to the great advances made in this science, the ability to solve complex real-life problems has increased [16]. After all, machine learning is able to deliver commercial and industrial value, being the ultimate purpose of a model to work properly in a production environment, a task that can prove to be far from simple. This follows the philosophy of Luigi Patruno, Senior of Data Science at 2U and founder of the MLinProduction blog [28], that states that no machine learning model is valuable, unless it's deployed to production. It is important for data scientists to keep in mind that production deployment and maintenance is an important process and that tools exist to handle these tasks. Keeping in mind the existence of MLOps can help and facilitate this process. Having a data scientist in the team with the required knowledge, will help to efficiently and effectively complete a machine learning project that works in a production environment.

1.3 Thesis Outline

The thesis outline provides a brief description of each of the sections that comprise this thesis. The document has been divided into a total of eight sections including the introduction to the subject. The structure of the report is as follows:

- Section 1: Introduction. Includes a brief description of MLOps and related topics such as Machine Learning (ML) and DevOps. In addition, the objectives and working methodology of the thesis are detailed along with the personal motivation for its development.

- Section 2: Previous Work. Collects a series of articles and reports associated with the topic in question, i.e., MLOps. These documents range from the year of origin of the topic to the present day. They are analyzed and explained in order to learn more about the context of MLOps.
- Section 3: Background. More detailed explanation of how DevOps and MLOps functionalities. The first of these concepts is considered the origin of MLOps, so it provides context to the main topic.
- Section 4: Specification & Design of the MLOps Framework. Introduces the proposed MLOps framework. The design of the framework is explained as well as the task and the dataset that the models must deal with.
- Section 5: Development of the MLOps Framework. Explains in detail how the MLOps framework has been developed, describing the architecture and operation of the code. The techniques used for preprocessing, model evaluation and training process, among others, are discussed.
- Section 6: Experimentation of the MLOps Framework. From the user's perspective, an example of use of the developed framework is shown. This helps to introduce the reader to the framework from a practical point of view. The main processes and actions are triggered to expose the framework's usefulness.
- Section 7: Conclusions & Assessment. Conclusion on the work done and objectives achieved. The proposed MLOps framework is also compared and evaluated.
- Section 8: Future Work. Lists some aspects of the developed framework that can be improved, extended or added as future work.

2 Previous Work

The main objective of this section is to select and analyze a number of papers related to the field of MLOps in order to provide context for the thesis. The reviewed literature is comprised between 2015 and 2022, taking into account that the first of these is considered the year of origin of MLOps. Having a selection of articles available over the years helps to represent the evolution of the literature in this field.

The first paper in relation to the field of MLOps is called "Hidden Technical Debt in Machine Learning Systems" and was published in 2015 by researchers at Google [31]. The goal of this paper is not to provide new functionalities, but to raise community awareness of the consequences of technical debt in machine learning systems. The concept is based on the metaphor introduced by Ward Cunningham in 1992, which helps to reason about the long term costs that occur by taking shortcuts in the development process. The authors mention that machine learning systems have a special capacity for incurring technical debt, since it has to deal with traditional code maintenance problems plus an additional set of ML-specific issues. They refer to this technical debt as 'hidden' since it is difficult to detect given that it often exists at the system level (data, processes, design) and not at the code level. Some examples of good practices provided in the paper are the use of modular system architectures, system testing and live monitoring of system behavior in real time, which is a critical aspect to achieve long-term reliability. The importance of this paper is enormous, since it is considered the origin of MLOps in the scientific literature. Overall, it seeks to raise awareness of technical debt in order to reduce as much as possible future problems in the maintenance of real-world machine learning systems.

"ModelDB: A System for Machine Learning Model Management" is a paper published in 2016 by authors from the Massachusetts Institute of Technology (MIT), in which they present an end-to-end system for the management of machine learning models called ModelDB [37]. As mentioned by the authors, ModelDB automatically tracks machine learning models in their native environment allowing data scientists to manage and explore models that have been built over time. The system architecture developed is based on three key components. The first component is a set of native client libraries for different machine learning environments (e.g. scikit-learn), which allows data scientists to experiment and build models with the machine learning environment of their choice. The second component is a front-end that allows visual exploration and analysis of models and pipelines. The third and last component is the back-end, which is based on a SQL database. This database can store all aspects of each machine learning experiment, including the pipeline and the models used. Overall, the paper proposes a MLOps framework, although the authors do not use this concept. This framework contains some of the key elements of MLOps such as monitoring, pipeline management and a visual interface for the user.

In 2018 a paper was published in relation with the release of MLFlow, one of the most popular open source platforms covering the needs of MLOps today. The paper is called "Accelerating the Machine Learning Lifecycle with MLFlow" and is written by members of the company called Databricks Inc [40]. The paper discusses the challenges around machine learning development and it also describes the MLFlow platform. As stated by the authors, this platform covers three key key ML development challenges: experimentation, reproducibility and model deployment. Most MLFlow features can be accessed through REST APIs, which can be called from any programming language. The main components of MLFlow are the tracking component (API for recording experiment runs, including code used, parameters, metrics, etc.), the project component (generic format for packaging code into reusable projects) and the model component (standardized format for packaging trained models which makes them easily deployable). The authors provide several examples of how MLflow has provided benefits in real-world scenarios, mentioning some of the companies and teams in the industry. In conclusion, this paper presents a platform that makes it easy for users to use their own machine learning algorithms, software libraries and development processes.

One of the most established formalizations of MLOps was introduced in 2019 and is referred to as Continuous Delivery for Machine Learning (CD4ML) [30]. CD4ML was proposed by ThoughtWorks and aims to automate the end-to-end lifecycle of machine learning applications. The authors define this approach as a software engineering approach in which a cross-functional team produces machine learning applications based on code, data and models in small and safe increments that can be reproduced and reliably released at any time, in short adaptation cycles. Short cycles means development cycles are in the order of days or even hours, not weeks, months or years. Automation of the process with quality built in is key to achieve this. The authors divide the CD4ML approach into three main steps. The first one consists of identifying and preparing the data for training, a task that is performed by data engineers. The second step consists of experimenting with different models until the candidate with the best performance is found, a task that is performed by data scientists. The third and last step consists of the deployment and use of the selected model in production, a task that is performed by the application developers. The following image shows the representation of the CD4ML end-to-end process:

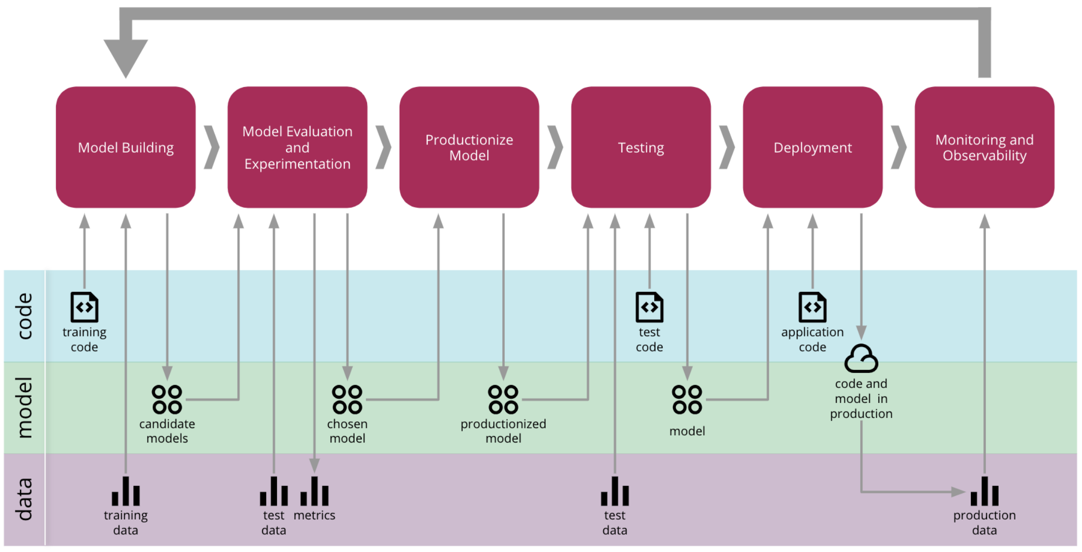


Figure 1: Continuous Delivery for Machine Learning end-to-end process. [30]

Due to the variations in machine learning methodologies, it is a challenge to generalize MLOps components and that is why numerous designs with different components have been proposed. Another popular design in MLOps, apart from the CD4ML discussed above, is the iterative-incremental process [39]. This MLOps process is divided into three phases: design, model development and operations. The first phase consists of identifying the potential user, designing the machine learning solution to solve his problem and evaluating the future development of the project. In the design of the machine learning solution where one of the tasks is to inspect the available data that will be necessary to train the model, it is fundamental to establish the requirements of the architecture that the machine learning application will have. The second phase iteratively executes different steps, such as data engineering and model engineering tasks, which aims to deliver a quality model to be executed in production. The third phase consists of a production deployment of the developed model, using DevOps practices such as testing, versioning, continuous delivery and monitoring. The following image shows the different phases of the iterative-incremental process in MLOps:

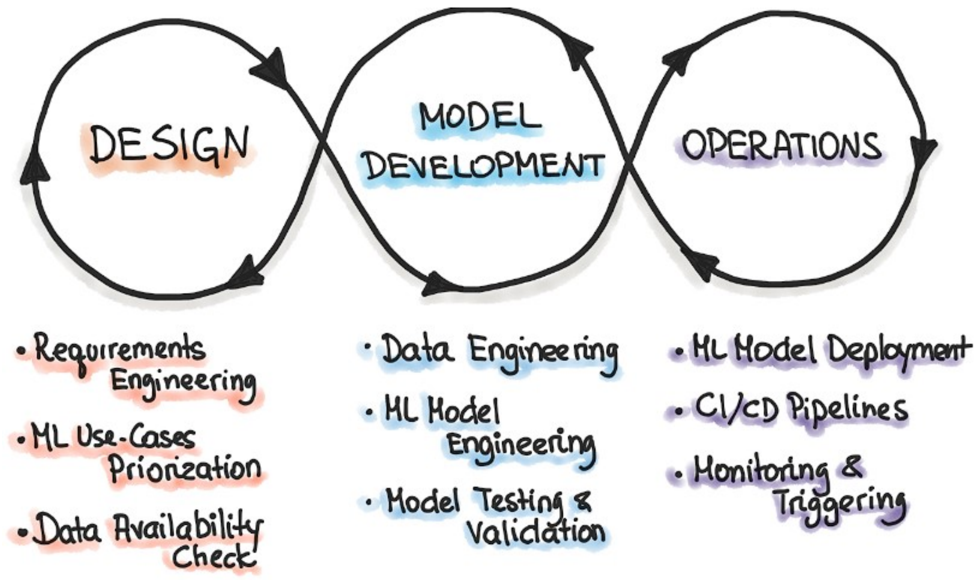

DESIGN
MODEL DEVELOPMENT
OPERATIONS
• Requirements Engineering
• ML Use-Cases Priorization
• Data Availability Check
• Data Engineering
• ML Model Engineering
• Model Testing & Validation
• ML Model Deployment
• CI/CD Pipelines
• Monitoring & Triggering

Figure 2: Iterative-incremental process in MLOps. [39]

“ModelOps: Cloud-based lifecycle management for reliable and trusted AI” is a paper published in 2019 by authors from the AI research group at IBM [20]. The paper proposes a cloud-based framework for managing the lifecycle of artificial intelligence applications in the cloud. The proposed platform is called ModelOps and allows for managing and executing model training and continuous learning pipelines while infusing trusted AI principles. The authors comment that the cloud plays a key role in bringing AI to the masses as they eliminate the need for owning expensive infrastructure to start experimenting with AI techniques. An important aspect of ModelOps is a language that allows for easy representation of the various artifacts involved in artificial intelligence (AI) solutions, such as datasets, model definitions, trained models, applications, monitoring events, algorithms, and platforms used for processing data, training models and deploying applications. All of these artifacts are versioned and their lineage is tracked for reproducibility and auditability. Another key component of ModelOps mentioned by the authors is flexibility, which allows for easy integration of proprietary model deployment platforms or customized data processing services into the ModelOps pipeline. Also thanks to this pluggable design it is possible to easily infuse quality control checks into the lifecycle of an AI application. In conclusion, the authors aim to enable users to evolve and continuously improve their AI models across the lifecycle, systematically reduce and manage the risks of model deployments, and ultimately create more reliable and trusted AI applications.

A paper called ”Continuous Training for Production ML in the Tensorflow Extended (TFX) Platform” was published in 2019 by researchers at Google [6]. The authors use Tensorflow Extended (TFX), one of the tools commonly used in MLOps frameworks, to implement continuous machine learning pipelines. These pipelines aim to keep the models continuously up-to-date with respect to the data. The tool is a Google-production-scale

ML platform based on TensorFlow. It provides a configuration framework and shared libraries to integrate pipeline components to define, launch and monitor machine learning systems. The article describes how continuous pipeline support has been implemented in the TFX platform and the main mechanisms to support this type of pipelines in production. Continuous pipelines need to maintain state in order to detect when new inputs appear and infer how they affect the generation of updated models. There are examples where deep learning models need to reinitialize the model weights from a previous run to avoid retraining on all the data accumulated up to that point. To manage this state, TFX has an ontology of artifacts which model the inputs and outputs of each component of the pipeline (data, models, statistics, etc.) and it also maintains the lineage between artifacts. Another key mechanism of TFX is orchestration, which allows asynchronous operation of components at different iteration intervals, enabling models to be produced as soon as possible. Finally, TFX employs several validation checks at different stages of the pipeline to prevent unsuitable models from being uploaded to production. These checks ensure that the models are trained on high-quality data (data validation), that they are at least equal to or better than the current production model (model validation) and that they are compatible with the deployment environment (serving infrastructure validation). In conclusion, the authors demonstrate the support capability around the continuous pipelines of one of the many MLOps tools, which is of interest since in the development of this thesis it is also desired to keep the models up-to-date with respect to data.

In 2020, authors belonging to the company VMware Inc. published a paper called "Challenges and Experiences with MLOps for Performance Diagnostics in Hybrid-Cloud Enterprise Software Deployments" [5]. The paper aims to explain how VMware addresses a number of challenges in operationalizing their machine learning-based performance diagnostics in enterprise hybrid-cloud environments. The main challenges facing the development of this ML system are the following: handling system performance drifts, feature engineering, model training, setting an appropriate alarm threshold and explainability. To tackle these challenges, the authors develop a fully automated MLOps pipeline that continuously trains new models based on the newly arrived batch of data. Models are maintained and version controlled in a model store. The pipeline evaluates a large number of feature sets by combining various preprocessing techniques and different ensembles of ML algorithms. In addition, continuous learning and deployment ensure that the model in production always learns the latest performance behaviors. In developing this pipeline, the authors conclude that they have learned a number of lessons, including the importance of monitoring and automation.

"CodeReef: an open platform for portable MLOps, reusable automation actions and reproducible benchmarking" is a paper published in 2020 that introduces an open MLOps platform that enables automated deployment of ML models across diverse systems in an efficient way [17]. For this reason, they refer to the platform as an MLOps cross-platform. The solution offered by CodeReef is a way to package and share models as non-virtualized, portable, customizable and reproducible archive files. As the authors say, such ML packages include JSON meta description of models with all dependencies, Python APIs, CLI actions and portable workflows necessary to automatically build, benchmark, test and customize models across diverse platforms, AI frameworks, libraries, compilers and datasets. With CodeReef, the user could create his own customized package with his own task to solve (object detection, object classification, etc.), dataset, model type, software (TensorFlow, PyTorch, MLFlow, etc.) and hardware (GPU, CPU, TPU, etc.)

used. The author's idea is to help researchers share their techniques as packages that are ready for production in order to compare different machine learning systems and select the most efficient ones. In conclusion, it is an initiative that demonstrates the evolution in the adaptability of MLOps frameworks, due to the fact that these are usually developed around a single task, dataset, software, etc.

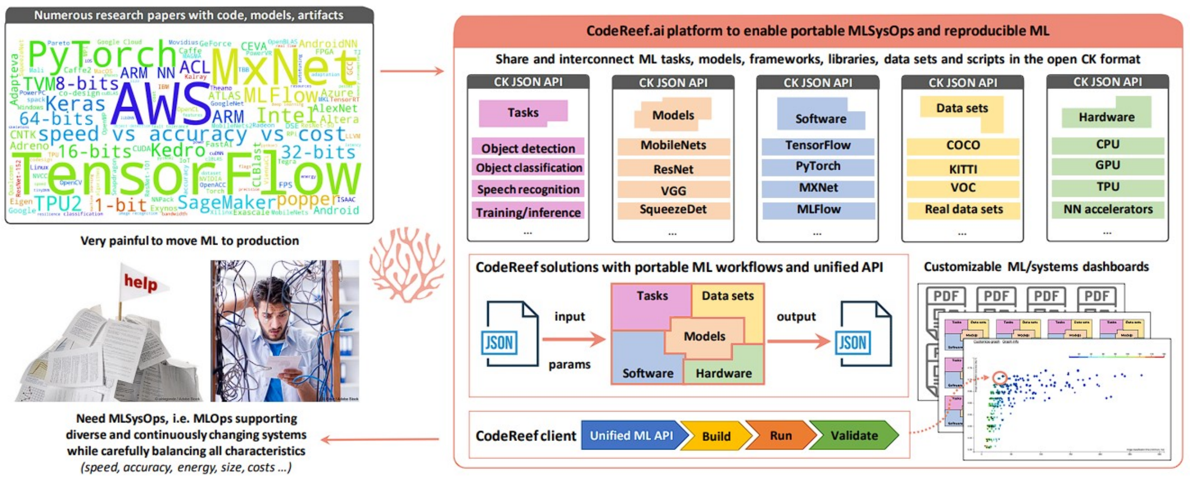

Numerous research papers with code, models, artifacts
Very painful to move ML to production
help
Need MLSysOps, i.e. MLOps supporting diverse and continuously changing systems while carefully balancing all characteristics (speed, accuracy, energy, size, costs ...)
CodeReef.ai platform to enable portable MLSysOps and reproducible ML
Share and interconnect ML tasks, models, frameworks, libraries, data sets and scripts in the open CK format
CK JSON API
Tasks
Object detection
Object classification
Speech recognition
Training/inference
Models
MobileNets
ResNet
VGG
SqueezeDet
Software
TensorFlow
PyTorch
MXNet
MLFlow
Data sets
COCO
KITTI
VOC
Real data sets
Hardware
CPU
GPU
TPU
NN accelerators
CodeReef solutions with portable ML workflows and unified API
input
params
output
JSON
Customizable ML/systems dashboards
PDF
CodeReef client
Unified ML API
Build
Run
Validate

Figure 3: CodeReef.ai: An open platform to keep track of ML systems research with portable workflows and reproducible crowd-benchmarking. [17]

In the year 2021 a paper is presented called "Towards MLOps: A Framework and Maturity Model", in which an MLOps framework is derived from a systematic review of the literature [21]. From this review, the authors extract insights to give an overview of the state-of-the-art of MLOps in practice. They mention that the MLOps systems in practice are divided into three parts, where the first part consists of the provision of data for the development of machine learning. The second part consists of the development of machine learning models where different experiments are run in order to optimize the selected model with hyperparameters and evaluate it to ensure that it fits the business case. The models and their information are saved in the model repository. The last part consists of the packaging, validation and deployment to production of the models, where once realized, the monitoring of its performance in production is important for its maintenance (retraining). The framework derived by the authors from this literature review consists of three pipelines: data, modeling and release. In addition, the paper also presents a maturity model (level of automation in each step) based on the analysis of different situations encountered in companies when adopting MLOps in practice. This model shows an evolution of the model towards a fully automated MLOps system, starting with automation of data collection, automation of model deployment, semi-automation of model monitoring and finally fully-automation of model monitoring. In conclusion, they present an interesting paper due to their previous literature review, as it means that the presented MLOps framework is based on generalized MLOps practices and therefore important.

A paper called "Edge MLOps: An Automation Framework for AIoT Applications" was presented in 2021 and aims to present an MLOps framework for artificial intelligence of things (AIoT) applications [29]. This framework is based on the concept of edge computing, which enables IoT devices to process data locally and not externally. The function of Edge MLOps is to automate machine learning at the edge, enabling continuous model training, deployment, delivery and monitoring. The authors mention that to achieve this they have synergized cloud and edge environments. Distributed edge devices can offer a number of advantages over a centralized cloud, as they can process data in real-time, minimize the need for communication and improve data privacy protection. On the other hand, they also have a number of disadvantages, such as low processing power or small storage capacity. For this reason, the authors decided to use a cloud platform to satisfy the computational requirements in different tasks like training models, storing data, etc. The edge device will handle other tasks such as continuous integration and data streaming with IoT devices and hyper-personalization. The authors performed a validation of their framework in a forecasting air quality situation, where the framework showed stability and also the ability to automatically retrain models when their performance deteriorated. In conclusion, this paper demonstrates the evolution of MLOps, in this case expanding into other fields such as the Internet of Things.

In 2022 a paper is presented on Jenkins, an open-source continuous integration tool that can be used to build pipelines to define and automate workflows in MLOps domain. It is titled “Jenkins Pipelines: A Novel Approach to Machine Learning Operations (MLOps)” [13]. The objective of the paper is to propose the design and implementation of pipelines for various stages of MLOps, namely data analysis, data preparation, training, testing and deployment on a single platform. Jenkins pipelines have a number of advantages, such as they can be easily edited and modified by the user. Another advantage is that even if the system where the pipeline is running loses power or shuts down, the pipelines can resume progress from where they left off. The key advantage according to the authors is that pipelines can be run in parallel on many systems simultaneously, which favors the speed of processes such as model preprocessing and model training. In addition, if for example the user decides to modify the data preparation stage, Jenkisn’s continuous integration (CI) pipeline would build and execute all the following stages automatically, which would considerably reduce time and manual efforts. Overall, the paper provides a tool for MLOps that is highly adaptable for use in other projects without the need for extensive modifications.

The last paper discussed in this section is called ”Towards MLOps in Mobile Development with a Plug-in Architecture for Data Analytics” and was published in 2022 [12]. The authors mention that both the use of mobile devices as IoT in industrial applications and the use of MLOps practices are increasing in practice. Despite this, support tools for MLOps are limited on mobile applications, mainly due to the unsuitability of native programming languages that must support machine learning tasks. The objective of the paper is to reduce this gap by means of a plug-in architecture to develop, deploy and execute machine learning modules on Android. The main feature is to allow adding new functionalities as plug-ins to the core application, providing extensibility, flexibility, customization and isolation of the application logic. In this architecture there are two core services that are composed of different plug-ins. The first one is called LoadDataService and is based on the collection of data from different sources (web, device, SQLite and files). The second one is called ProcessDataService and is in charge of executing data processing and analysis tasks, being able to invoke machine learning inference scripts as a way of executing Python code. Thanks to this, the ML engineer only has to upload the ML components (Python script, trained model, data labels) without modifying the plug-in part, which is written in Java. The authors conclude that this system provides benefits such as modularity, extensibility, customization and parallel development. In conclusion, this work is an indicator of the evolution of MLOps as it seeks to improve its expansion to other IoT devices such as smartphones.

Finally, as a summary, a table of all the scientific papers and articles mentioned in this section is presented in terms of the problem they wish to address and what they propose to solve it.

Table 1: Summary of the reviewed scientific papers and articles related to MLOps.

Paper	Adressed Problem	Proposal
[31]	Hidden technical debt in machine learning systems.	Raise awareness of the problem and provide advice.
[37]	Model building is ad-hoc. No practical way to manage models that are built over time.	ModelDB, a novel end-to-end system for management of ML models.
[40]	Machine learning development creates new challenges: experimentation, reproducibility and model deployment.	Implementation of MLflow, an open source platform. It uses generic APIs that work with any ML library, algorithm and programming language.
[30]	Developing, deploying and continuously improving ML applications is more complex compared to traditional software.	Introduction of CD4ML, the discipline bringing continuous delivery practices to ML.
[39]	Lack of established best practices and tools for MLOps components.	Proposal of the iterative-incremental process design of an MLOps system.
[20]	New challenges when moving ad-hoc ML experiments to automated operationalized pipelines.	ModelOps, a cloud-base framework and platform for end-to-end development and lifecycle management of AI applications.
[6]	Disruptions in continuous ML pipelines can increase model staleness and degrade the quality of the derived services from these models.	Implementation of the main mechanisms of TensorFlow Extended (TFX) for supporting continuous ML pipelines.
[5]	Operationalizing ML solutions is challenging across the industry.	Presents how VMware addresses a set of challenges in operationalizing an ML application in an hybrid-cloud environment.
[17]	Research and advances in MLOps are not unified due to the diversity of methodologies in ML.	CodeReef: Open MLOps platform that enables automated deployment of ML models across diverse systems. It offers a way to package and share models.
[21]	The adoption of MLOps in practice is still in its infancy and there are few common guidelines on how to effectively integrate it.	An MLOps framework and a maturity model derived from a literature review.
[29]	Traditional ML in IoT devices has limitations. ML models are often trained centrally and deployed manually which reduces scalability.	Edge MLOps framework for automating ML in IoT devices, enabling model training, deployment, delivery and monitoring.
[13]	Developing a ML lifecycle is very complex and takes a lot of time and manual effort.	Jenkins: Open-source continuous integration tool used to build pipelines to define/automate workflows in MLOps.
[12]	In mobile app development there is limited tooling support for MLOps.	A plug-in architecture for developing, deploying and running ML modules for data analytics on the Android platform.

In conclusion, it can be noticed how different papers in the list share the same objectives but provide different solutions, reflecting the dispersion of the community in the evolution of MLOps. The paper proposing CodeReef [17] is an example of the attempt of unification in MLOps advances. Although research continues to propose different architectures, frameworks and pipelines with the same intent, each offers different benefits and drawbacks, making them particular in their own way. With this in mind, despite not having a unified way of building MLOps frameworks, all the advances provide their own contribution that helps in the advancement of the field.

3 Background

This section provides an overview of DevOps and MLOps to know in more detail about how these sets of practices work. As mentioned in the introduction, DevOps is the main basis of MLOps, which is why a detailed explanation of DevOps is included to serve as context.

3.1 DevOps

In an article that provides an overview of recent DevOps technologies at the time of publication [14], Christofer Ebert, one of its authors, provides a brief and interesting definition of DevOps: "DevOps is about fast, flexible development and provisioning business processes. It efficiently integrates development, delivery, and operations, thus facilitating a lean, fluid connection of these traditionally separated silos". Overall, DevOps is a set of practices that tries to bridge the gap between development and operational teams through automation. For instance, miscommunication problems between team members are avoided, which accelerates problem resolution. In addition, cross-functional teamwork and automation improves the speed, optimization and quality of software delivery. Automation is key to provide quality deliveries with short cycle time.

Automating the software delivery process ensures continuous delivery and feedback loop of the software. Development, testing and deployment processes are part of the continuous delivery, which combines all of them into one streamlined operation. The main goal of DevOps is to speed up this entire process through automation. In this way, users will avoid excessively long waits for the software release cycle and will be able to test and give feedback on the software faster. All DevOps core processes are defined in a workflow, which in turn defines a set of tools and practices. It defines the processes from development to maintenance, helping teams to build, test and deploy software quickly and efficiently [34].

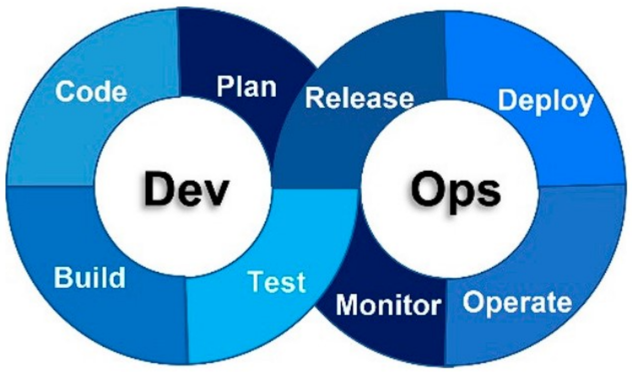


Figure 4: Typical DevOps workflow. [34]

The DevOps workflow is composed of the following phases in sequence: planning, coding, building, testing, releasing, deploying, operating and monitoring. The workflow starts with the planning phase, where the project lifecycle to be developed is planned, defining the requirements and creating an initial execution plan. In the next phase, i.e., the coding phase, the developers write the code in accordance with the requirements established in the planning phase [11]. One of the important features of this phase that is not always implemented is code versioning. This makes the software easier to maintain and helps with workflow automation. The building phase involves the packaging of the entire software (business logic, software dependencies and the environment) and the guarantee of the code integrity maintenance by evaluating the correctness and quality of the software. In the case of a new code modification, a rebuild of the project is performed at this stage.

The workflow continues with the testing phase, where continuous automated testing is performed to ensure the quality of the software and that it executes as planned. This phase is also used to find problems or bugs that may arise in certain circumstances. After testing comes the releasing phase, where a final check of the code is performed to verify that it is ready to enter a production environment. Once the approval has been given, it proceeds to deployment, the phase in which the software finally goes into production. It is a process that can be automated to achieve a continuous deploying or redeploying of the software. The next phase is the operation phase, which focuses on the maintenance and testing of the application in the production environment, ensuring system reliability and high availability [34]. The last phase consists of monitoring the application in order to have a control of its performance and thus be able to plan the next iteration of the workflow.

These components do not operate individually in practice, but are connected sequentially in a so-called pipeline. A DevOps pipeline is a set of processes that are automated and tools that assist in the operation of these processes. Each pipeline is unique depending on the characteristics of the application, but the most common elements in a pipeline are: continuous integration (CI), continuous delivery (CD), continuous deployment and continuous monitoring [34]. These pipeline elements contain the DevOps components explained previously and can be combined to form larger pipelines or work as pipelines on their own (e.g. continuous monitoring).

Continuous integration (CI) is the practice of integrating code changes into existing code base so that any conflicts between different code changes made by developers are quickly identified and solved [18]. The building and testing phase are DevOps components that are involved in continuous integration, since after each code commit, automated code building is performed, followed by the corresponding automated tests. As the planning and coding phases are carried out before the building phase, they can be considered to be related to continuous integration. CI prepares new code changes for deployment, this practice is therefore critical to increase deployment efficiency.

Continuous delivery (CD) involves the release phase of DevOps, since its objective is to keep the application ready for deployment to the production environment. It is common for continuous deployment to be executed later, as it automates the deployment of the code. If this element is not available, the deployment to production would be performed manually by the user. Continuous deployment is not usual, as it is only useful when a large number of developers work together to provide numerous releases every day [18]. Whether with or without the use of continuous deployment, these pipeline elements follow continuous integration, forming a larger pipeline called CI/CD. A schematic of this pipeline is shown in Figure 5.

Lastly, another important pipeline element is continuous monitoring (CM), which automatically monitors the performance and health of the deployed application in production [34]. CM also helps to detect complications while the application works in production. It comes in at the end of the DevOps pipeline.

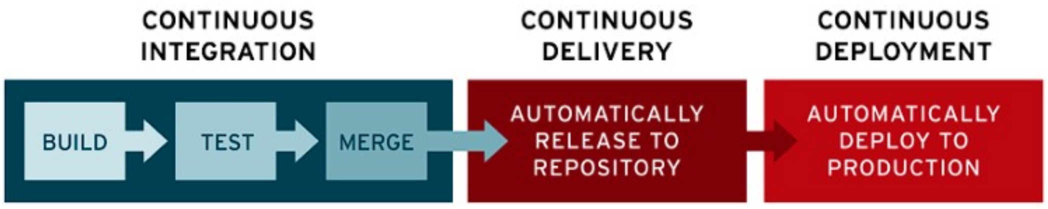


Figure 5: CI/CD pipeline with continuous deployment. [19]

3.2 MLOps

In most real-world applications, data is constantly changing, implying that In order to run models in production environments, it is valuable to keep them updated and maintained. It is therefore vital that the efforts and times for their deployment and maintenance are reduced as much as possible, which is some of the benefits that MLOps provides. As explained in the introduction section, MLOps is an extension of the DevOps methodology that seeks to provide automation and monitoring at all steps of the ML lifecycle, which refers to the complete process of developing, deploying and maintaining the ML application. It also offers communication and collaboration between data scientists while automating and productizing machine learning algorithms. DevOps has had a major impact on MLOps, where the main links between them are the concepts of continuous integration (CI) and continuous delivery (CD), which allows software to be produced in short cycles, ensuring that it can be reliably released at any time [36].

The idea of MLOps is to adapt DevOps concepts to be implemented in the ML lifecycle development. However, there are DevOps tools and pipelines that cannot be applied to machine learning systems because they must work with other software assets, which causes MLOps to add additional steps in its process. Due to variations in machine learning methodologies, it is difficult to generalize the components of MLOps workflow. As mentioned in the previous work section, designs with different components have been proposed, such as iterative-incremental processes [39] and Continuous Delivery for Machine Learning (CD4ML) [30]. The following figure shows a generic MLOps workflow in relation to the DevOps components previously explained.

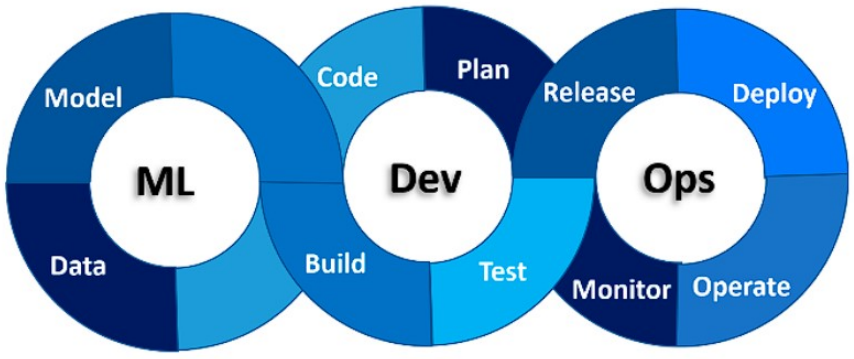


Figure 6: Generic MLOps workflow in relation with DevOps components. [34]

Compared to the DevOps workflow shown in Figure 4, model and data appear as two new components. In addition, the existing components such as testing, deployment and monitoring are different from those used in the DevOps workflow [34]. In addition, it should be noted that most of the steps in MLOps, unlike DevOps, are experimental and may vary during implementation, e.g. hyperparameter optimization. As in the DevOps workflow, the MLOps workflow starts with the planning and coding processes. The objective of these components are the same, to define the initial requirements and code based on them. In the iterative-incremental process [39], this part of the workflow would be in the design phase.

The next component is unique to the MLOps workflow. It is related to data and can be divided into four sub-processes: data extraction, data validation, data analysis and data preparation [34]. The data extraction process is the first to be carried out. It consists of collecting data from different sources, e.g. from files such as CSV, or also from a data warehouse in the cloud, among other sources. Extracting data properly can avoid problems in the future, since, for example, in the case of classification tasks, care must be taken to ensure that the data is balanced. To make sure that the data is adequate, the following process called data validation is carried out, where data quality problems are detected and solved. Some common problems that may be encountered are the presence of null values or problems related to the data type, among others. The next process is data analysis, where the objective of the data scientists is to understand the characteristics of the data, which is vital for the subsequent model building and feature engineering process. Finally, data preparation is performed, where one of the main actions consists of splitting the data into three different datasets: training, test and validation. The feature selection is also performed and extra features are added if deemed appropriate in the analysis, in addition to fixing quality problems that may remain to be solved. Once the three datasets are prepared, they are sent to the model.

The next phase in the MLOps workflow is related to the model. In this stage, the data scientists define the structure of the model to be used to subsequently perform the model training and model validation operations. In the model training phase, several experiments are performed in order to optimize the selection of hyperparameters, i.e. the model is trained once or several times with different variable values and with the same training dataset. In each of these experiments, model evaluation and model validation is performed. The former returns the performance measurement of the model based on the use of the validation dataset, which allows the comparison between different experiments performed with the model [34]. Model validation consists of validating the predictions of the new model by using the old or new data and comparing it with the predictions of the old model.

Once the desired model or models have been developed, we move on to the build and testing phase. The building process shares the same objective as in DevOps, to perform a packaging of the software involved in the machine learning system. The next process executed is testing, which is important to ensure the quality of software systems supported by machine learning. Although researchers have applied some traditional software testing concepts, there are a number of challenges in testing ML-based systems that make these techniques ineffective [24]. In the testing process of the machine learning system, unit and integration tests related to the convergence of models and other variables related to them must be performed. Also the testing of the code that involves the development of the model and the format of the data in the input and output is performed.

The next phases are the release and deployment of the machine learning application. The releasing phase, as in the DevOps workflow, consists of performing a final check of the application to verify that it is ready to go into production. On the other hand, the deployment of a machine learning application is not straightforward as in DevOps, since the retraining and deployment of models can be triggered by the arrival of new data [34]. To perform these actions, an automated pipeline is created, which will evaluate the performance of the models over time to decide whether they have to be retrained and deployed. Triggers for automated model training and deployment can be calendar events, messaging, monitoring events, as well as changes on data, model training code, and application code [39].

Finally, there are the operation and monitoring phases. The operation process, in the same way as in DevOps, is focused on maintaining the machine learning application in production, detecting and resolving the appearance of problems to ensure its reliability. In the monitoring process, the performance of existing machine learning models working on the same dataset is controlled. Through the monitoring feedback, it is possible to perform the required triggers to retrain the models, in order to keep them updated when new data arrives or when necessary.

The MLOps pipeline involves the workflow components explained above and can be divided into three main elements: continuous integration (CI), continuous deployment (CD), continuous training (CT) and continuous monitoring (CM). The continuous integration (CI) in machine learning is based on the building and testing components. It consists of the validation of the data and models that are being added, as well as unit tests and integration tests of the surrounding code. This automatic integration must be triggered when adding new data or models, including when adding or modifying code. It is common to apply this pipeline in a versioned and reproducible way, since with version control it is possible to make a comparison in production [25]. The continuous deployment (CD) element involves the release and deployment of components. It consists of verifying the compatibility of the models with the production environment as they evolve and then deploying them in production, all in an automated manner.

As in DevOps, it is common to refer to CI and CD elements as the same general CI/CD pipeline. In the case of machine learning, they are complemented by a pipeline element specific to MLOps called continuous training (CT). This element consists of automatically retraining the machine learning model in order to keep it updated as the data changes. Retraining will be triggered whenever a condition is met, such as a deterioration in performance or the entry of new data. In the latter case, there are multiple ways to decide the timing of model training: incremental training (training with new data as the data comes in), batch training (training once a significant amount of new data is available) and retraining (retraining the model from scratch once a significant amount of data is available) [38]. The continuous training (CT) element involves the data and model components, so it also includes the processes of data extraction, validation and preparation, as well as model evaluation and model validation.

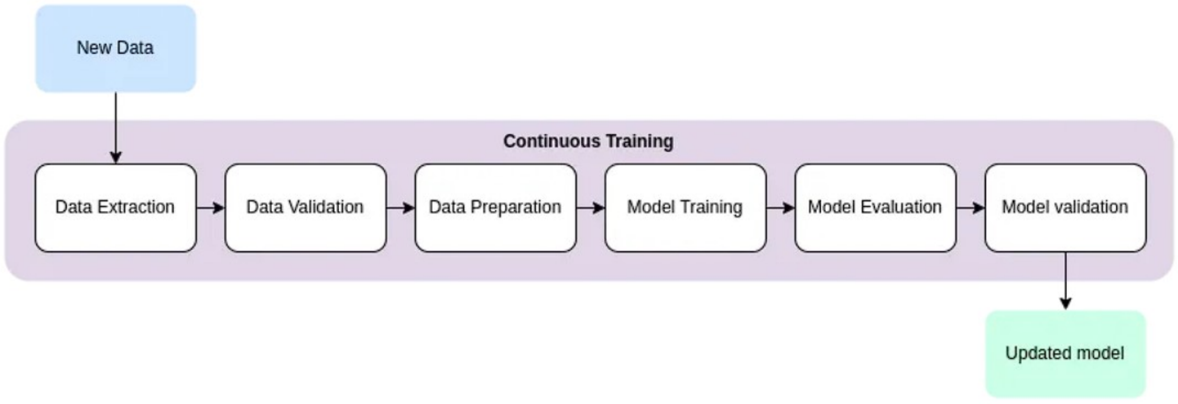


Figure 7: Processes of Continuous Training (CT). [38]

Finally, in MLOps it is essential the element of continuous monitoring (CM), which is in charge of monitoring the production model to control that it performs as expected. This is a vital process since model performance is affected by the input of new data over time. Continuous monitoring (CM) includes the operate and monitor components of the MLOps workflow, so, in addition to monitoring model performance, it also monitors possible risks that may affect the correct functioning of the system. The monitoring can help, for instance, in the detection of inconsistency problems and errors in the data or also in the detection of concept drift. The latter case occurs when the statistical properties of the target variable vary over time, causing the predictions to be less accurate [25]. Continuous monitoring (CM) may be a necessary complement to continuous training (CT) when the retraining trigger is based on the performance decay of the model. Below is a schematic of the retraining triggered by performance decay, in which retraining of the model is performed when the F-1 score is below a specified threshold.

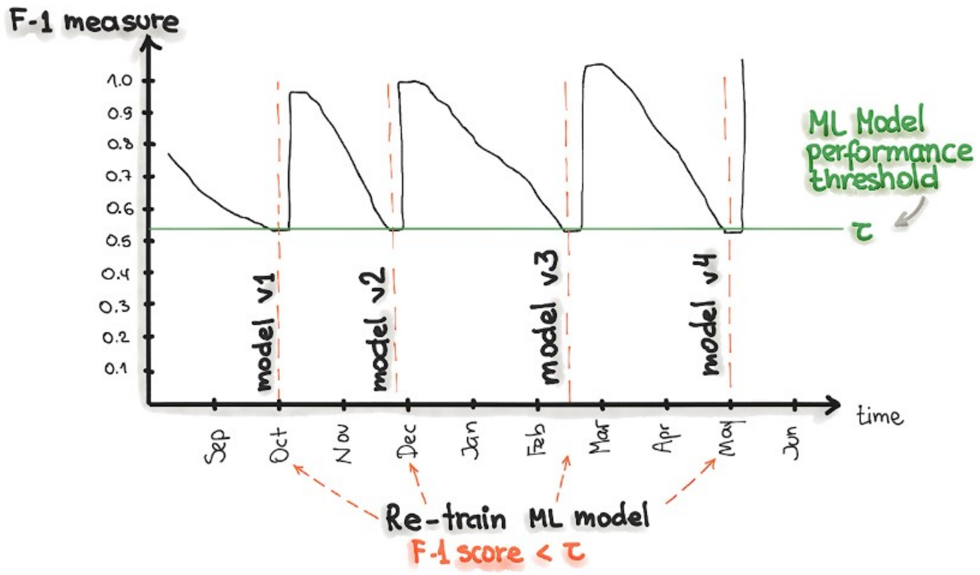


Figure 8: Machine Learning Model Decay Monitoring. [39]

Not all elements of the pipeline need to be implemented, they can be selected depending on the requirements of the desired machine learning system. The more elements are included in the pipeline, the higher level of automation is achieved. The level of automation determines the maturity of the machine learning process. Google cloud proposes three levels of maturity for MLOps processes [10], starting from the initial level with manual training and deployment, up to running the entire pipeline fully automatically.

- Level 0 of MLOps is the traditional data science process, which is experimental and iterative in nature. All steps of the pipeline are executed manually, i.e., from data preparation to model training and validation. This process focuses only on the implementation of the trained model as a prediction service.
- Level 1 of MLOps includes the execution of the model training automatically, i.e. the continuous training (CT) element is included. When new data becomes available, the training process will be triggered. The data and model validation processes are also included in an automated way. Continuous monitoring (CM) can be included in this level of automation, as it can be vital to monitor the performance and status of the models over time.
- Level 2 of MLOps, i.e. the last level, introduces the CI/CD pipeline for a complete automation of the system. It includes the elements of continuous integration, deployment, training and monitoring. This level of maturity achieves fast and reliable model deployments in production due to the automation. The system automatically builds, tests and deploys the data, model and training pipeline components.

In conclusion, MLOps facilitates the entire machine learning procedure related to the deployment and maintenance of models in a production environment by avoiding development and operations bottlenecks. In this section, components of MLOps workflow, elements of the MLOps pipeline and levels at which the MLOps system can be automated have been described. These are the pillars of MLOps, which can also include other important features and functionalities such as versioning of data, models and code, registry to store trained models or model metadata store.

4 Specification & Design of the MLOps Framework

In the previous sections different works and papers on MLOps over the years have been provided, as well as a more detailed introduction to this set of tools that is evolving over time and is increasingly used. This theoretical introduction serves as the basis of the project, where the ultimate goal is the development and design of an MLOps framework based on the knowledge acquired. The design and specification of the developed MLOps framework in question is explained below along with the related task to be solved by the models and the used dataset.

4.1 Task & Dataset

The MLOps framework developed is based on the maintenance and management of models that perform a regression task. Specifically, the models work with a dataset called 'M5 Forecasting-Accuracy' which is a time series. The dataset stores the number of units sold of a series of different Walmart products in different states of USA for each day from 2011 to 2016. Specifically, these are products from four different stores in the state of California, three stores in the state of Texas and three stores in the state of Wisconsin. The products are categorized into three different groups: hobbies, foods and households. Below is a diagram that facilitates the explanation of the dataset:

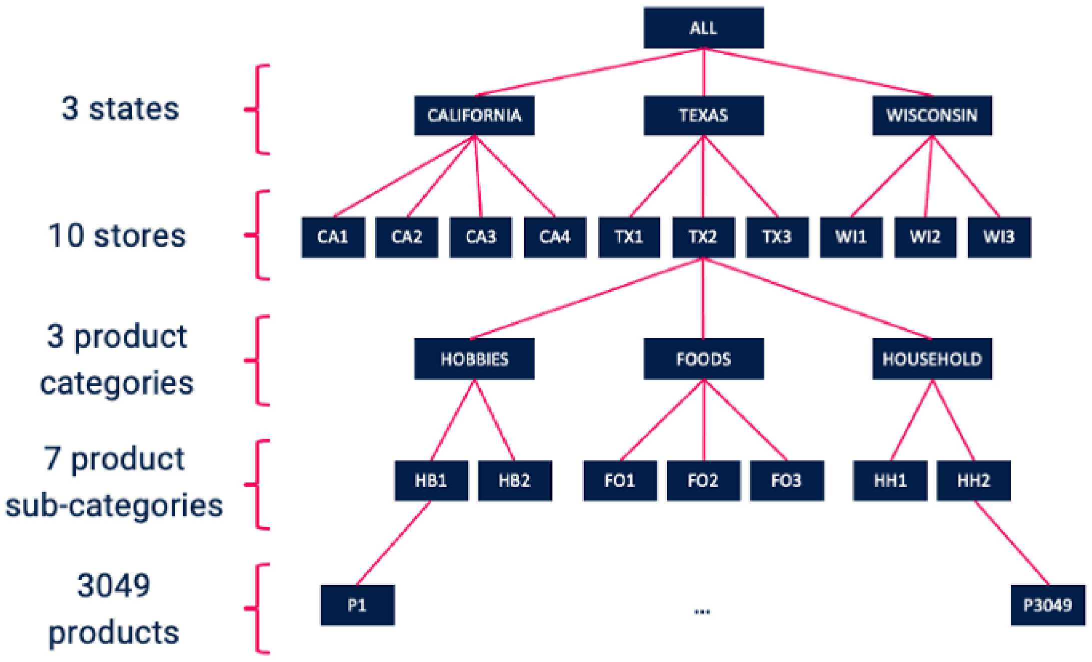


Figure 9: Structure of the M5 Forecasting-Accuracy Dataset. [22]

Due to the large number of records in this dataset, it has been decided to reduce it in order to reduce the computation time significantly. In this way, the different tasks performed in the MLOps framework can be executed faster. Thanks to this, it has been possible to perform the tests more effectively during development, in addition to facilitating the exemplification of the MLOps system. In order to reduce the dataset, two measures have been taken, the first one is to use only the records of the first Walmart store in each of the US states. The second measure consists of covering the years between 2013 and 2016, discarding the first two years which are 2011 and 2012.

The task consists of estimating the point forecasts of the unit sales of various products sold. Therefore, the target variable is the unit of products sold for each store in each state. Specifically, the objective is to predict sales for the next 28 days. To achieve this, the models must perform a regression task. The task specifications were set by the University of Nicosia through the Kaggle website [26], in the form of a competition with a financial prize that was published in 2020. To participate in the competition, the authors offer a number of 5 csv files with the necessary data to perform the exercise. In the case of this project, the necessary information offered by each of the files has been gathered in a general dataset that has been used to work with.

The objective of the project is to develop an MLOps framework around this task. This framework will facilitate a number of processes related to the machine learning lifecycle, such as model training and evaluation. Through automation and the dashboard used as a user interface, greater control and efficiency is gained over the use of several models in a common task. It should be noted that existing MLOps frameworks are usually focused on a single task, since they seek the maintenance, management and monitoring of models working on the same task. The use of the same framework for models that work in different tasks does not fulfill its purpose, which is to deploy the most appropriate model to work in production, where the task is unique. Anyway, as mentioned in the previous work section, there are advances in technologies that allow the adaptability of MLOps frameworks in different tasks (not simultaneously), as is the case of CodeReef [17], which facilitates the customization of a framework depending on the task, dataset and models used among other features.

4.2 MLOps Framework Design

In the background section it has been commented that there are three maturity levels proposed by Google cloud [10], which are indicators of the level of automation of an MLOps framework. The framework proposed in this project coincides with the characteristics of maturity level 1, where continuous training (CT), model evaluation and continuous monitoring (CM) are available, all of them being automated processes. On the other hand, there is no CI/CD pipeline like in level 2, which is composed of continuous integration (CI) and continuous deployment (CD) elements. With this pipeline, a complete automation of the framework would be achieved, which would allow a fast and reliable deployment of models in production. In the case of the project there is no production environment, so this functionality has not been added. Instead, the framework simulates a complete pre-production environment, where all existing models are maintained and controlled. In a real case, where a production environment is available, one of these models would be automatically selected from the pre-production environment for deployment, following a specific condition to select the most suitable one.

In the proposed framework, a functionality has been added that consists in the manual execution of certain processes, such as training and evaluation of the models. This manual execution can be ordered by the user when necessary, providing greater control over the system. In general, an MLOps framework has been developed that can work in an automated way once a series of models are available, but avoiding a closed automated pipeline by adding a functionality that allows user intervention. This framework can be divided into three main blocks: data generation and preparation, orchestration and dashboard. Each of these blocks deals with certain functionalities that in general make up the complete framework. The following is a detailed explanation of what the blocks consist of.

4.2.1 Data Generation & Preparation

The data generation and preparation block involves two main tasks, the first one being the generation of new data batches, and the second one the preprocessing of the data taking into account the new batches added. This system basically consists of simulating the growth of a dataset with new data over time, assimilating it to a real case. Due to its characteristics, MLOps offers certain advantages in these cases, since the models will vary their performance over time due to the growth of the data. The maintenance and monitoring of the models is vital to work in a production environment. The original M5 dataset is static, so new data is not being generated. To show the usefulness of the developed MLOps framework, it has been decided to simulate the input of new data. This will make it possible to appreciate the change in the performance of the models as they are monitored.

The data generation and preparation process it is automated to run every day. The first task that this process performs is the generation of a new batch of data, which corresponds to the next day of the dataset. In this case, the last day for which there is a record is May 22, 2016, so the next batch of data that will be created will correspond to May 23, 2016. The values of this new batch of data are generated with certain conditions, which are different for each of the features of the dataset. The new data is not real, but the conditions are intended to ensure that the generated data does not differ significantly from the pattern of the original data.

The second and last task performed by this block is the preprocessing of the data and its splitting in the test and train set, so that it can be used for model evaluation and training. For this purpose, a preprocessing pipeline has been created that must be executed with the entire dataset. Each time a new batch of data is added, the batch is not preprocessed and then added, but first added to the dataset and then preprocessed. The time involved is clearly longer, but due to some preprocessing techniques, it requires all the information that the dataset provides, which does not allow to preprocess only one day's batch of data. Once the data has been preprocessed, the last 28 days of the dataset are selected to create the test data, since the objective is to make a prediction of the sales units of the products in the next 28 days. The rest of the days make up the training data. Below is a diagram summarizing the pipeline of the data generation and preparation block:

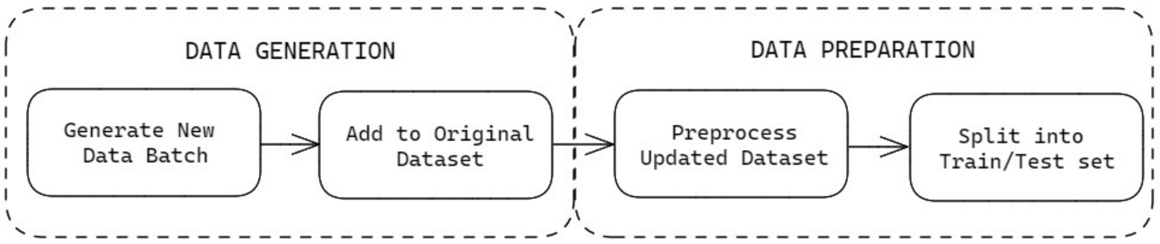


Figure 10: Data generation and preparation pipeline.

Later in this report (section 5), it will be explained in more detail how the feature values of the new data are decided, and which techniques the preprocessing pipeline uses.

4.2.2 Orchestration

The orchestration module is the main element of the MLOps framework, since it provides the necessary tools to automate the desired tasks. Thanks to this element, it is possible to implement the continuous monitoring process (CM), as well as the continuous training process (CT). Additionally, it is responsible for scheduling the execution of the data generation and preparation pipeline, which is useful to simulate the entry of new data over time. Due to the decision to offer manual control to the user over most of the tasks that are performed automatically, the orchestration module will be ready to receive orders instantaneously regardless of the time the task is scheduled.

For the development of this project, it has been chosen to use Apache Airflow as the orchestration module of the framework. It is an open-source tool that allows the orchestration of workflows or pipelines. In addition, it has several functionalities that provide useful information to the user, e.g. about the progress of the execution, the status of the scheduled workflow, the logs and the code behind the workflow. Apache Airflow also allows manual triggering of tasks that are scheduled for execution, functionality that has been considered for implementation in the developed framework.

In Airflow, workflows are represented as DAGs (Directed Acyclic Graph), which are composed of several tasks. A DAG consists of a python file built with a specific structure. The first step in building this file would be the definition of tasks, which are the basic unit of execution of Airflow. In total, there are three basic types of tasks [2]:

- Operators, predefined task templates that can be used to build most parts of the DAGs.
- Sensors, a special subclass of operators that waits for an external event to happen.
- TaskFlow-decorated @task, a custom Python function packaged as a task.

When defining the DAG for a series of tasks, the order in which they are executed must be indicated. This is known as the relationships between tasks, which define how they relate to each other, i.e., their dependencies. Once the execution of all the tasks has been completed in the established order, the workflow defined by the DAG will have finished its execution. As an example, below is a figure representing a directed acyclic graph composed of a series of interrelated tasks.

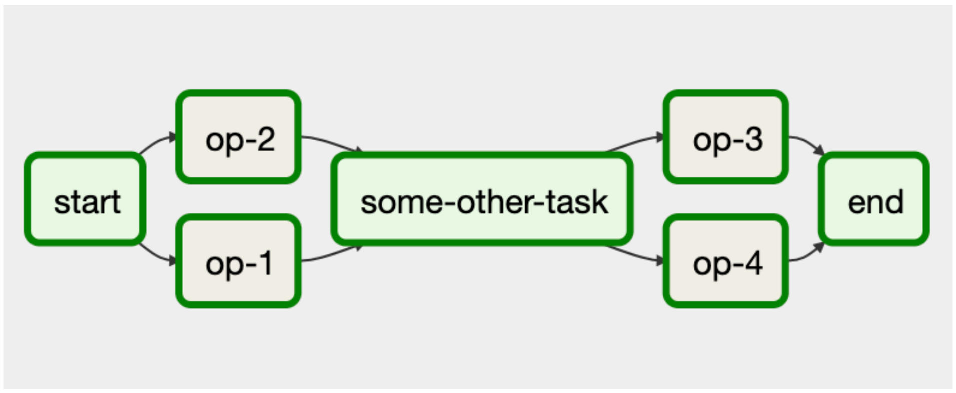
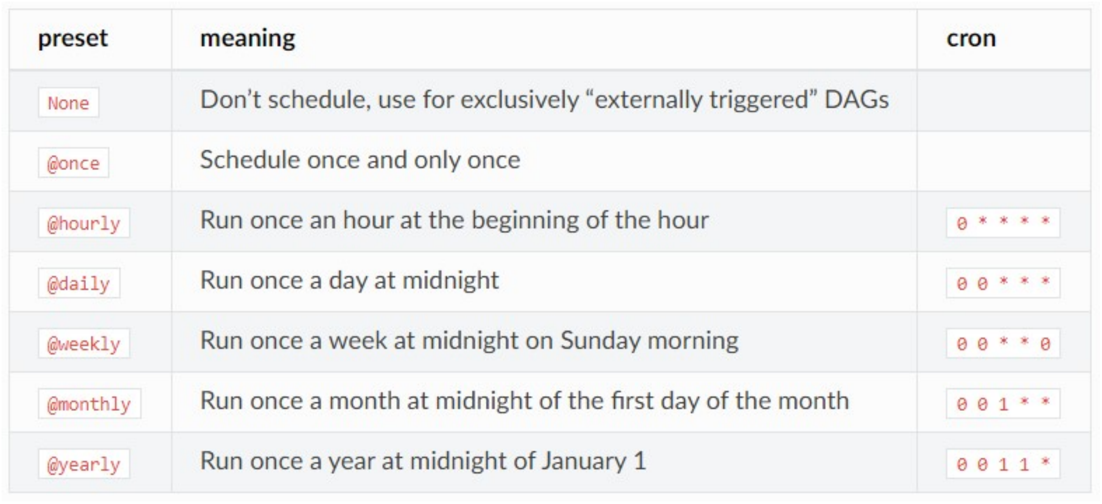


Figure 11: Example of a group of related tasks forming a DAG. [9]

In this figure, the tasks are represented with rectangles and the relationships between them are represented with arrows. The execution flow goes from left to right, where the task ’op-1’ and ’op-2’ would be the first to be executed in parallel, since they are the tasks connected to the beginning of the DAG. The next task to be executed is ’some-other-task’, which will not be executed until the two previous tasks finish, since it is dependent on both of them. Once ‘some-other-task’ is finished, the execution of ’op-3’ and ’op-4’ will start in parallel, which will mark the end of the DAG.

Airflow allows scheduling DAGs to be executed every certain period of time, which can be decided by the user through the use of cron expressions. A cron expression consists of a coded string describing the details of the schedule, i.e. how often the DAG will be executed and at what time of the day. This is not the only method to define the scheduling of the DAGs, since the user can also use the cron ’preset’ or even a datetime.timedelta object. Below is a summary table of the cron expressions and cron ’presets’ available in Airflow:



preset	meaning	cron
`None`	Don’t schedule, use for exclusively “externally triggered” DAGs	
`@once`	Schedule once and only once	
`@hourly`	Run once an hour at the beginning of the hour	`0 * * * *`
`@daily`	Run once a day at midnight	`0 0 * * *`
`@weekly`	Run once a week at midnight on Sunday morning	`0 0 * * 0`
`@monthly`	Run once a month at midnight of the first day of the month	`0 0 1 * *`
`@yearly`	Run once a year at midnight of January 1	`0 0 1 1 *`

Figure 12: Available cron expressions in Apache Airflow. [1]

Summarizing, DAGs can be defined by three main components: tasks, task relationships and task scheduling. These elements make up the structure of the python file used to create the DAG. In section 5, the different DAGs generated and how they have been defined will be explained in more detail.

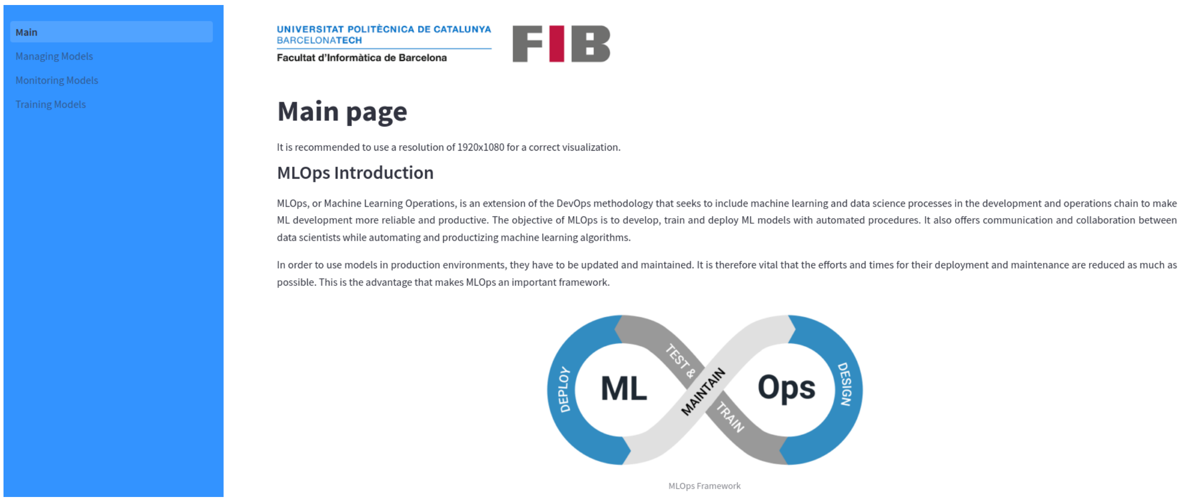
Finally, the Airflow installation must be performed in order to implement it as the project orchestration module. There are different ways to do it, but a procedure for learning and exploration has been chosen (recommended by the official Airflow website). The disadvantage of this installation method is that the adaptation for the use of real-world situations can be complicated, but that is not included in the project scope. This installation method involves the use of a platform known as Docker, which is an open source platform that allows developers to build, test and deploy applications quickly [3]. Its functionality consists of packaging software in units called containers, which have the necessary software to run, such as libraries, code, tools, etc. Docker is similar to a virtual machine which virtualizes the server hardware, only that containers virtualize a server’s operating system. Airflow can be installed using Docker Compose, which is a Docker plug-in that allows the definition and customization of application services through a YAML file. Upon installation, Airflow can be run and started through a series of simple commands.

4.2.3 Dashboard

The dashboard is the user interface of the MLOps framework. Airflow also offers an UI, but the goal is to focus the entire framework on a single interface capable of obtaining information from different files stored in the Python project and being able to interact with the orchestrator in turn. In general, there will be an interface where the information and actions needed to manage and monitor the machine learning models are provided.

The dashboard is designed using Streamlit, an open-source Python library that allows developers to build user interfaces quickly and effectively. Basically, it turns data scripts into shareable web apps, all using Python. Streamlit provides a number of functions that allows the developer to add different elements such as tables, graphs, messages among others [33]. Adding these types of elements is as simple as declaring a variable, so there is no need to write a back-end, define routes, write HTML, CSS, JavaScript, etc.

The interface will be divided into four pages with which the user will be able to consult information and interact: Main, Managing Models, Monitoring Models and Training Models. The first page, called Main, consists of a summary of the proposed MLOps framework. It contains some of the information discussed in this section 4, i.e., the type of task solved by the models, the dataset used and the three main blocks that make up the designed framework.


Main
Managing Models
Monitoring Models
Training Models
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona
FIB

Main page

It is recommended to use a resolution of 1920x1080 for a correct visualization.

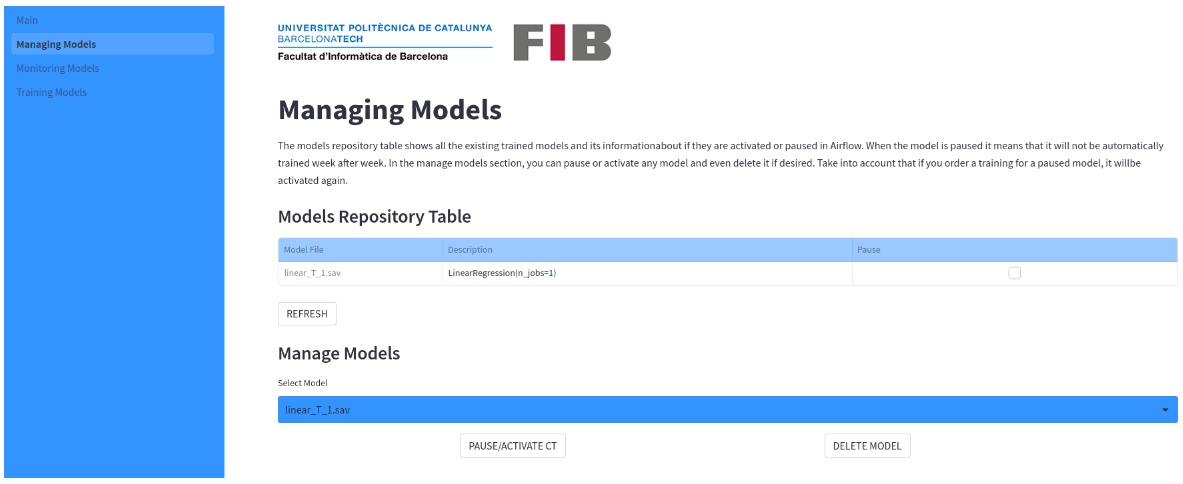
MLOps Introduction

MLOps, or Machine Learning Operations, is an extension of the DevOps methodology that seeks to include machine learning and data science processes in the development and operations chain to make ML development more reliable and productive. The objective of MLOps is to develop, train and deploy ML models with automated procedures. It also offers communication and collaboration between data scientists while automating and productizing machine learning algorithms.

In order to use models in production environments, they have to be updated and maintained. It is therefore vital that the efforts and times for their deployment and maintenance are reduced as much as possible. This is the advantage that makes MLOps an important framework.

Figure 13: MLOps introduction in Main page.

The next page is called Managing Models, which offers two actions with respect to existing models. The user can consult the 'Model Repository Table' to find out which models exist in the system and whether they are paused or active. If the model is active, it means that it will be automatically retrained according to the schedule. If the model is paused, the automatic retraining will not be performed until it is active again. The first action offered in the Managing Page, allows the user to pause or activate the continuous training (CT) of any existing model in the system. The second action removes the model from the system, deleting the DAG in Airflow and the corresponding records in Python. Both actions can be activated by using two different buttons: Activate/Pause CT and Delete Model. The user can select the model to which the action is to be applied from a list of possible selections. Each action requires Airflow communication in order to make a request. Furthermore, by pressing either button, the user will be kept informed of the status of the process through a series of messages. In addition, a button is added to refresh the information provided in the table, which can help to update the information after having executed an action. The following figure shows the page layout:


Main
Managing Models
Monitoring Models
Training Models
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Managing Models

The models repository table shows all the existing trained models and its informationabout if they are activated or paused in Airflow. When the model is paused it means that it will not be automatically trained week after week. In the manage models section, you can pause or activate any model and even delete it if desired. Take into account that if you order a training for a paused model, it willbe activated again.

Models Repository Table

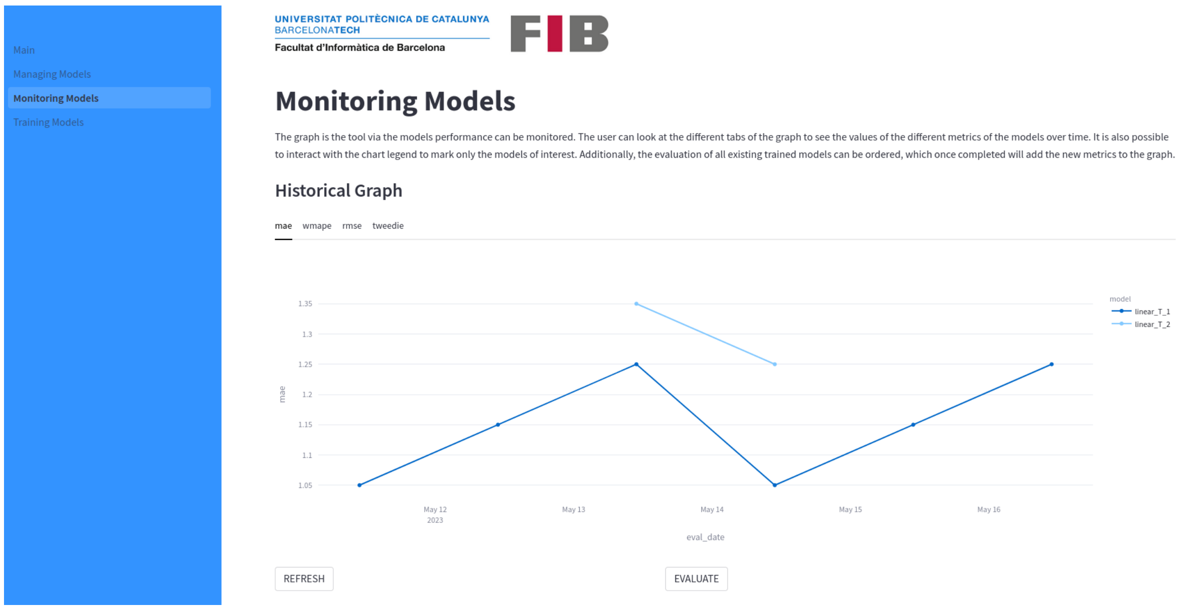
Model File	Description	Pause
linear_T_1.sav	LinearRegression(n_jobs=1)	☐

Manage Models

Select Model: linear_T_1.sav

Figure 14: Managing page.

The Monitoring Models page is vital for visualizing the performance of the models over time. The main component is an overall visual space that is divided into four graphs, one of each for a different metric calculated when evaluating the models. Through these graphs it is possible to observe the value of the metrics for each evaluation, training or retraining that has been performed on the models over time. It has an interactive legend where all models are listed, in which the user can select those they want to visualize in the graph. This page has two buttons, one of them is used to refresh the information of the graph, while the remaining is used to execute an evaluation process of the models. When activating this last button, a trigger of the correspondent DAG in Airflow is performed, which is in charge of evaluating and obtaining the metrics of each of the existing models in the system. At the start of the system execution, the user will be informed about the status of the process through the use of messages. The following figure shows the page layout:


Main
Managing Models
Monitoring Models
Training Models
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Monitoring Models

The graph is the tool via the models performance can be monitored. The user can look at the different tabs of the graph to see the values of the different metrics of the models over time. It is also possible to interact with the chart legend to mark only the models of interest. Additionally, the evaluation of all existing trained models can be ordered, which once completed will add the new metrics to the graph.

Historical Graph

Figure 15: Monitoring page.

Finally, the Training Models page is where the user can train new models or retrain previously generated models. On this page there is the 'Trained Models Table', where the information of the last record of each of the existing models is displayed. A record of a model is generated when it is evaluated, trained for the first time or retrained. In this table the user can see when was the date of its last training and its last evaluation, together with the metrics obtained and the hyperparameters of the model. The table can be refreshed by activating the 'Refresh' button, useful when an action has been previously performed on any model.

In the 'Train Your Model' section, the user can select the type of model and its hyperparameters. In the case of this framework, a limited number of model types from scikit-learn (Python library) are available, namely three different ones: linear regression, decision tree and gradient boosting. For each of them the user can select the desired value for different hyperparameters, which are also limited. The ranges of values for each hyperparameter and the hyperparameters available have been decided on the basis of their importance, justifying this choice. The selected hyperparameters for each type of model are as follows:

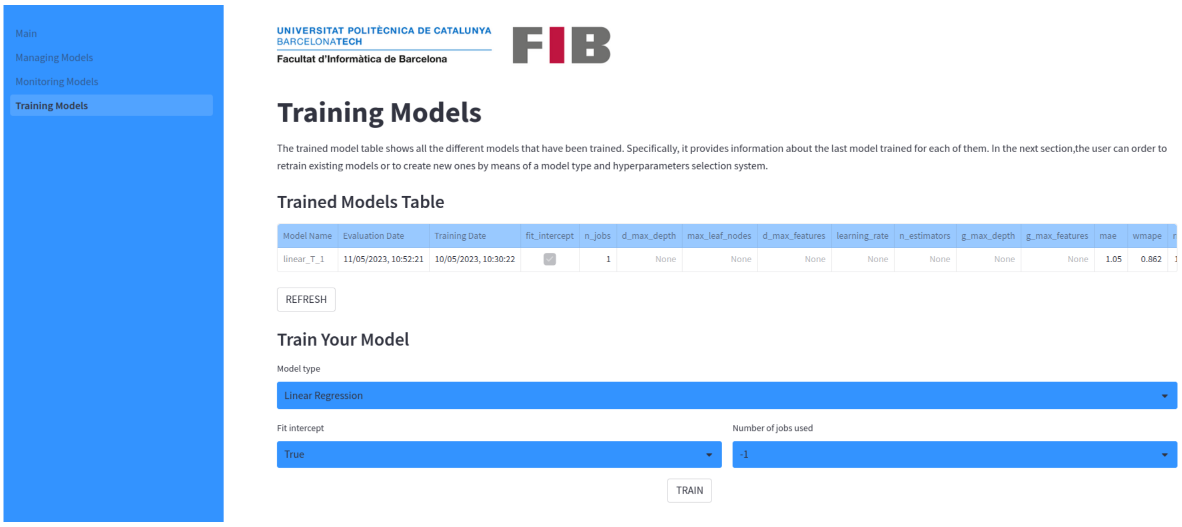
- Linear Regression: fit_intercept and n_jobs.
- Decision Tree: max_depth, max_leaf_nodes and max_features.
- Gradient Boosting: learning_rate, n_estimators, max_depth and max_features.

In the case of a linear regression model, there is a small number of hyperparameters which are not usually used for fine tuning and performance improvement, but rather to indicate the modus operandi and configuration of the model. For this reason, two parameters have been selected that can be useful for the user.

In the case of decision trees and gradient boosting, there is a larger number of useful hyperparameters to perform fine tuning, so some of them have been selected. In both cases, hyperparameters have been chosen to help reduce the computation time. This type of hyperparameters allows to limit the maximum depth of the tree, the maximum number of features considered to perform a split, etc. It is worth remembering that the main objective of the development of this MLOps framework is to show the usefulness of this tool, so it is not focused on achieving the best performance of the models trained on the task. For this reason, selecting hyperparameters that allow to reduce the computation time is appropriate in this case, as it allows experimentation to be faster and efficient.

Finally, two of the hyperparameters selected for the gradient boosting model affect the boosting operation of the model (learning_rate and n_estimators), while the other two affect each tree individually (max_depth and max_features). In this case, all hyperparameters except learning rate have been selected in order to have the possibility of reducing the computation time. The learning rate hyperparameter has been added additionally as it is a great option to improve the performance in the fine tuning of the model.

The user can train the desired model by selecting the type model and values of its hyperparameters. Once selected, the 'Train' button must be activated to train the model. In case the model does not exist in the system, a DAG will be created in Airflow corresponding to that model, which means that it will be trained every certain period of time automatically (continuous training). In addition, a first training and evaluation is performed to know its performance. When a model is generated, it will be stored in the Python model repository with a coded name, such as 'linear_T_1'. This name indicates the type of model followed by the value of its hyperparameters. In case the model already exists, a retraining of the model would be ordered by making a request to Airflow to trigger the corresponding DAG. As in the rest of the pages, during the execution of the training, the user will be kept informed of its status through the use of messages. The following figure shows the page layout:



Training Models

The trained model table shows all the different models that have been trained. Specifically, it provides information about the last model trained for each of them. In the next section,the user can order to retrain existing models or to create new ones by means of a model type and hyperparameters selection system.

Trained Models Table

Model Name	Evaluation Date	Training Date	fit_intercept	n_jobs	d_max_depth	max_leaf_nodes	d_max_features	learning_rate	n_estimators	g_max_depth	g_max_features	mae	wmape	r
linear_T_1	11/05/2023, 10:52:21	10/05/2023, 10:30:22	✓	1	None	None	None	None	None	None	None	1.05	0.862	1

Train Your Model

Model type: Linear Regression

Fit intercept: True

Number of jobs used: -1

Figure 16: Training page.

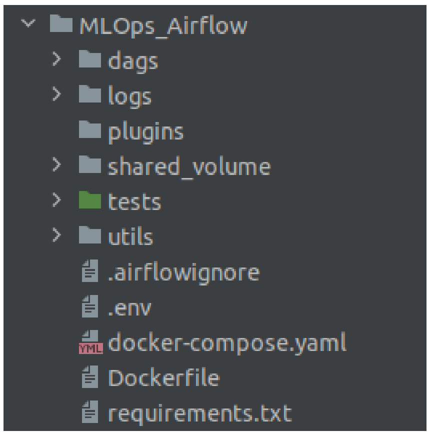
The back-end processes that are carried out with each action on each of the pages will be explained in more detail in section 5.

5 Development of the MLOps Framework

A superficial explanation of the MLOps framework developed has been made in the previous section, with the purpose of giving the reader a general idea of how it has been designed and which functionalities have been implemented. In the current section, we will go into more detail regarding the operation of the code developed in the different blocks that comprise the framework: data generation and preparation, orchestration and dashboard. Techniques used in the preprocessing, the definition of DAGs and the back-end processes that the user can initiate through the dashboard are detailed. Prior to that, a brief clarification of the project architecture is made, highlighting the main folders and files.

5.1 Project Architecture & Content

The project has been developed in Python, where on one side there is the part corresponding to Airflow and on the other side the part corresponding to the dashboard. Therefore, the project is structured in two main folders: MLOps_Airflow and MLOps_Frontend. First, the MLOps_Airflow folder, where the Airflow installation is performed by using Docker Compose, is discussed. Next, the contents of this folder can be observed:



```
MLOps_Airflow
  dags
  logs
  plugins
  shared_volume
  tests
  utils
  .airflowignore
  .env
  docker-compose.yaml
  Dockerfile
  requirements.txt
```

Figure 17: Structure and content of the MLOps_Airflow folder.

The MLOps_Airflow folder has content that comes from the Airflow installation. The 'dags', 'logs' and 'plugins' folders are the main folders in relation to the orchestrator. As developers, files have been generated in the 'dags' folder, as this is the location where the DAGs that will be executed periodically by Airflow should be stored. Both the 'logs' and 'plugins' folders are not used, since the former is where the Airflow execution logs are generated, and the latter is where plugins are added, which have not been required in the case of this project. Other files that appear with the installation are the '.env' and 'docker-compose.yaml' files, fundamental to define the Airflow services (docker-compose.yaml) and to define the correct Airflow user (.env), which must match the host user to avoid permission conflicts.

On the other hand, the MLOps_Airflow folder is composed of additional folders and files that have been developed manually. On the one hand, there are the 'Dockerfile' and 'requirements.txt' files that have the purpose of adding dependencies to Airflow in order to execute the tasks defined in the DAGs. The '.airflowignore' file is created to indicate to Airflow the folders that it should ignore, since the orchestrator only needs to access the ‘dags’, ‘logs’ and ‘plugins’ folders. The ignored folders correspond to those generated manually for the development of the framework: ’shared_volume’, ’test’ and ’utils’. On the one hand, the ’test’ and ’utils’ folders are complementary because the framework can work without them. These folders, respectively, contain files to perform tests (unittest) of some of the functions related to the DAGs and the initial versions of the data among other files that are not used. On the other hand, the ‘shared_volume’ could be considered the core folder of the project. Its contents are shown below:

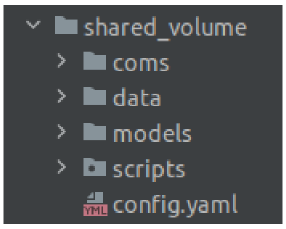


Figure 18: Structure and content of the shared_volume folder.

The ’shared_volume’ folder is the location where the dashboard related code and the orchestrator related code access to read or write required information. In other words, it is considered to be the intermediary folder between the two parties. The following is a brief explanation of the contents of this folder, as it will be important to explain the development of the framework:

- The ’coms’ folder contains shell files in charge of making requests to Airflow, and that its execution can be ordered from the dashboard by the user. In this same folder, ’.json’ files are generated to store the status of the process or the result of the executions performed, which are also accessed from the dashboard to inform the user through the use of messages.
- The ’data’ folder contains the ’.csv’ files that store the data used by the models, i.e. the dataset, the preprocessed data, the train set, the test set, etc. There is also an important file named ’historical_dataset.csv’, which stores the information about the training and evaluations of the models over time.
- The ’models’ folder is the model repository of the framework, where each model is stored once it has been generated for the first time by the user from the dashboard. The models stored in the repository will be accessed by the orchestrator for the execution of the DAGs.
- The ’scripts’ folder stores a series of python files related to the process of generating new DAGs when the user creates a new model, which will be explained in detail later.
- The ’config.yaml’ file stores a series of paths used in the code related to the orchestrator and dashboard, making the project more adaptable by allowing paths to be changed directly throughout the project.

Next, the contents of the remaining main folder named MLOps_Frontend are detailed. MLOps_Frontend is responsible for all the dashboard functionality. It contains the code that organizes the elements of the different dashboard pages, such as tables showing important information or graphs showing the metrics of the models performances. It also provides buttons to order different tasks such as training or evaluating a model, and informs the user about the state of the running task. The main files in this folder are

Python files and each of them corresponds to one of the dashboard pages explained in the previous section. The following image shows these files and the rest of the folder contents:

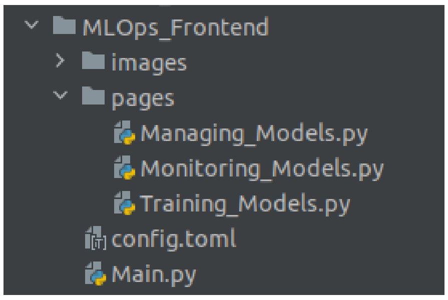


Figure 19: Structure and content of the MLOps_Frontend folder.

The 'images' folder contains a series of images used in the 'Main' page, where a brief introduction of the framework is made in order to inform the user. The 'config.toml' file has the function of defining the dashboard design in terms of text font, primary color, secondary color, etc. The rest of the files correspond to the different pages of the dashboard and except for the 'Main' page, they are all located in the 'pages' folder. It is required to leave the 'Main' page out of the folder, since it is the page with which the dashboard is initialized through a command. By initializing it this way, the rest of the files inside the 'pages' folder will be automatically detected as different pages inside the dashboard.

5.2 Data Generation & Preparation

The data generation and preparation block is considered the first block of the designed MLOps pipeline. As previously explained, the models work with a dataset called M5 of time series type, which is static. In order to simulate the growth of the dataset, a data generation and preparation system has been developed. This simulation helps to visualize more effectively the change in model performance over time. This process is defined as an Airflow DAG, since it is scheduled to be executed periodically. A diagram of the tasks that make up the DAG is shown below:

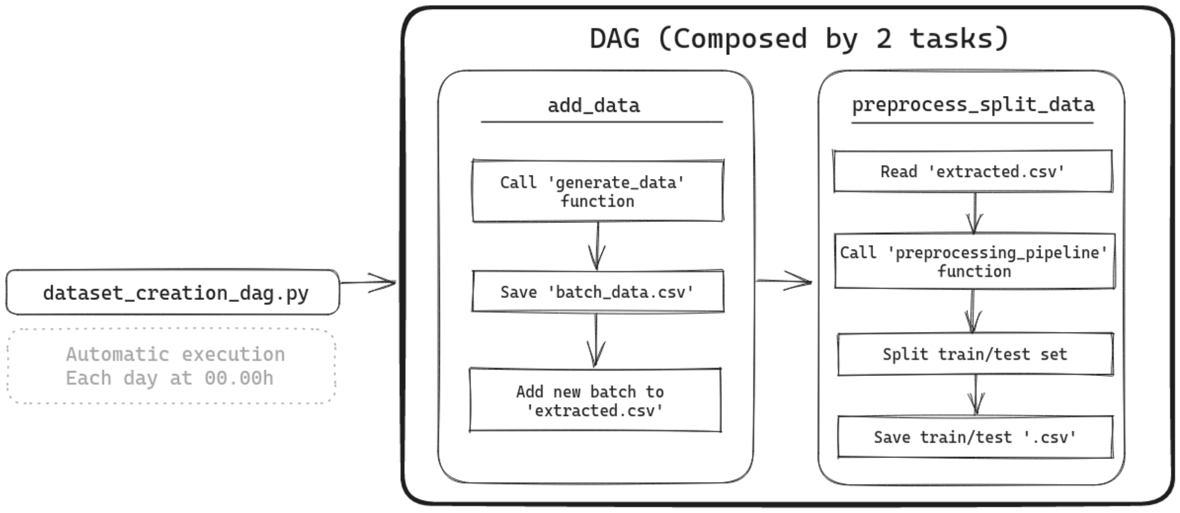

DAG (Composed by 2 tasks)
add_data
Call 'generate_data' function
Save 'batch_data.csv'
Add new batch to 'extracted.csv'
preprocess_split_data
Read 'extracted.csv'
Call 'preprocessing_pipeline' function
Split train/test set
Save train/test '.csv'
dataset_creation_dag.py
Automatic execution Each day at 00.00h

Figure 20: Definition of the DAG for data generation and data preparation.

The DAG in charge of the execution of the data generation and preparation process is composed of two tasks. The first one consists of the generation of new data, where the function called 'generate_data' is the main process. This function is developed in a python file inside the 'scripts' folder, which in turn is located in the 'dags' folder. This 'scripts' folder contains a series of python files with functions used in the different DAGs defined, among them, the 'preprocessing_pipeline' function used in the second task of the DAG.

The 'generate_data' function generates a new batch of data corresponding to the next day of the dataset, where each row corresponds to a different product from Walmart stores. To begin with, a complementary '.csv' file was created containing all the rows corresponding to the last recorded day of the dataset. From this file, the function generates the corresponding data for the next day following a series of norms for each of the existing features. A summary table of these norms for each of the features of the dataset is shown below:

Table 2: Summary of feature norms used to generate new values.

Feature	Description	Norm
'id' related	Define product id, store and state among others.	Remains the same.
'd'	Count of days recorded in the dataset.	Add 1 to the previous batch.
'sales'	Unit of sales of the product on that day.	Select the number of units sold randomly according to a probability. The probability is obtained using the previous batch of data.
'date'	Date of sale of the products.	Add one day to the previous date.
'wm_yr_wk'	Count of weeks recorded in the dataset.	Add 1 every seven days.
'wday'	Day of the week indicator using a range from 1 to 7.	Add 1 with respect to the previous batch. Back to 1 when the previous batch is 7 (maximum).
'weekday'	Name of the day of the week.	Select the week name that corresponds to the new number indicated by 'wday'.
'month'	Corresponding number of the month.	Extract the month number from the 'date' feature.
'year'	Corresponding number of the year.	Extract the year number from the 'date' feature.
'event' related	Indicates 1 or 2 events corresponding to each day of the year, e.g. New Year's Eve, Christmas, etc. The type of event is also indicated.	From a dictionary where the event and its corresponding day are stored, obtain the event from the day that is being generated.
'snap' related	3 binary flags for whether the stores in each state allowed purchases with SNAP food stamps at this date (1) or not (0)	From a dictionary where the binary flags and its corresponding day are stored, obtain the snap flags from the day that is being generated.
'sell_price'	The store and item IDs together with the sales price of the item as a weekly average.	Choose a number at random that has a difference of -0.1/+0.1 with respect to the value of the previous batch, only when the week is over (wday=0).

Once the 'generate_data' function is executed, it returns the new batch generated from the rules indicated in the previous table. This batch is saved in the 'batch_data.csv' file, performing an overwrite of the previous batch stored, which corresponds to the previous day of the dataset. The last step of the first DAG task is the addition of the new batch in the 'extracted.csv' file, updating the general dataset with the new batch generated.

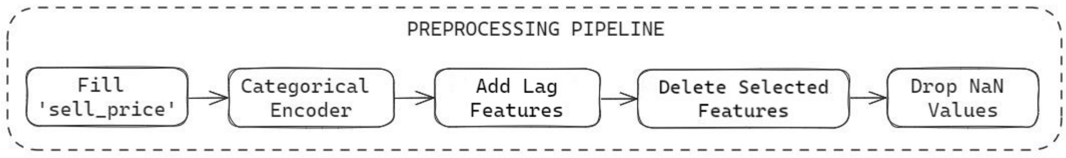


Figure 21: Definition of the DAG for data generation and data preparation.

In the first stage of the preprocessing pipeline, a fill method is performed on the 'sell_price' feature, which indicates the average sales value of a product in a given week. The 'sell_price' feature contains several missing values, which it has been decided should be replaced by numeric values avoiding the elimination of these rows, since it would mean a considerable loss of data and information (this feature has more than 19% of NaN values). To perform the fill, it has been decided to carry out a backward fill method without taking into account the possible special prices on event days. A backward fill is necessary because missing values are only found in the first recorded days of each of the existing products. With the backward fill, it is ensured that all NaNs disappear. The event-related features also have a high ratio of missing values, which have been replaced by the 'NoEvent' category. In this way, when performing the backward fill, only those rows where the event category is 'NoEvent' are selected. For days with events, the NaN values are replaced by the mean of the correspondent product price.

The second stage, called 'Categorical Encoder', converts the values of categorical features into numerical values by using encoding techniques. Only the features that will finally be kept in the dataset are considered, avoiding the preprocessing of features that will be discarded in the 'Delete Selected Features' stage. The first action carried out at this stage is not categorical encoding, but the unification of the three snap-related features by using logical operators, thus reducing the total number of features. The first categorical encoding technique carried out is with respect to the 'item_id' feature, by which the original ids of the different items are converted into numerical ids.

The rest of categorical features have been differentiated into two different types, those with a low number of categories and those with a high number of categories. Only the features that will finally be kept in the dataset are considered, avoiding the preprocessing of features that will be discarded in the 'Select Important Features' stage. The features with a low number of categories are: 'cat_id', 'state_id', 'event_type_1' and 'event_type_2'. In this case, the one hot encoding technique is applied, where for each different category, a new column with binary values is generated. The features related to the event type are two in total, since two different events can coexist on the same day. These two features can share the same category (e.g. religious or cultural), thus generating two new columns (event_type_1_Religious and event_type_2_Religious). In these cases, the columns will be unified into a single one by means of a logical operator 'or', getting a single column for each category that exists in both features. The only exception is the 'NoEvent' category, which will be unified by an 'and' operator, since both features must indicate that there is no event on that day.

The features with a high number of categories are: 'event_name_1' and 'event_name_2'. In these cases the target encoding technique is applied., which is the process of replacing the categorical value with the mean of the target variable, i.e., the 'sales' feature. At this stage of the DAG the data is not divided into train and test data, so the target encoding cannot be applied to all the data, since the future train set would be collecting information from the test set. To avoid this, the separation is made (taking the last 28 days for the test set) to apply target encoding on both sets and join them again afterwards.

In the next stage of the preprocessing pipeline, a series of lag features are added to the dataset. Lag features correspond to past values of the target value, in this case, past values of product sales units. They are useful in time series cases, since information about product sales in previous days can help in training the model. In this stage called 'Add Lag Features', five lag features are added with the sales of 5, 10, 15, 20 and 28 days ago. Two features indicating the mean sales of the last 7 and 14 days of the corresponding product are also added. Finally, apart from the lag features, it is decided to create a new binary feature in the dataset that indicates whether it is a weekend or not. This step is easily adaptable, since the user can intervene to change the number of past days on which the lag features and mean lag features are created, as well as deciding whether to create the feature indicating the weekend. Adding only 1 or 2 day lag features could also be a good option for training models.

The last two stages of the pipeline are called 'Delete Selected Features' and 'Drop NaN Values'. The first one consists of the elimination of a series of features that are selected according to the importance of the information they provide. In the case of this dataset, the features that provide the same information as others ('id', 'dept_id', 'store_id' and 'weekday') or that increase sequentially over time ('d', 'wm_yr_wk' and 'year') are deleted. In the first case, the features related to the id of the products are deleted because it has been decided to represent the id with the features 'item_id', 'cat_id' and 'state_id', which already provide the necessary information to distinguish between products. As for the last stage of the pipeline, it consists of eliminating the possible remaining missing values from the dataset, namely those that have appeared after adding the lag features.

These are all the stages that make up the preprocessing pipeline, which once executed, the DAG task will split it into the train and test dataset. The split is simple, the last 28 days of the dataset comprise the test set, while the rest of the days will form the train set. The reason why the split is performed in this way is because the task to be performed by the models is the prediction of sales for the next 28 days. Since the test set is data that the model has not seen in its training, it is useful to evaluate the performance of the model when making the prediction. Once the split is done, both sets are saved in '.csv' files.

5.3 Orchestration

Airflow is the orchestrator of the developed framework, and is in charge of the programming and execution of the existing DAGs. In the previous section, the DAG corresponding to the generation and preparation of new data has been explained, while in this section the DAGs corresponding to the evaluation and training of models are the main topic. In addition, it is also detailed how the execution of the DAGs has been programmed and for what purpose, including the one corresponding to the creation of new data.

5.3.1 Evaluation DAG

The evaluation DAG forms the process of predicting the models of the number of sales of the products in the future 28 days. Specifically, the evaluation of the existing models in the model repository of the project, located in the 'models' folder within 'shared_volume', is performed. Each of the models performs its prediction and then a series of metrics are computed: MAE, RMSE, WMAPE and Tweedie. A diagram of the tasks that make up the DAG is shown below:

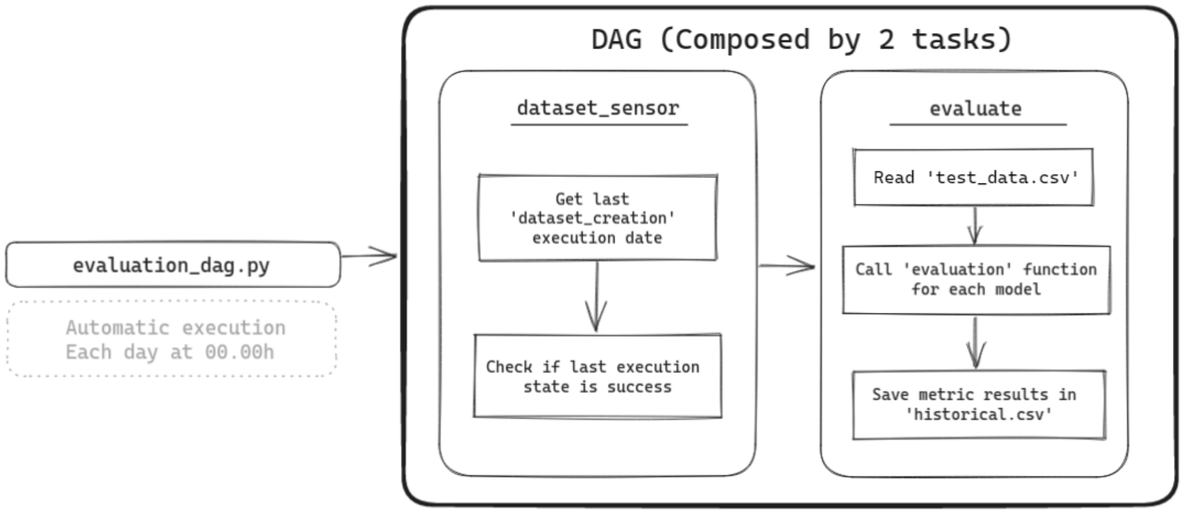

DAG (Composed by 2 tasks)
evaluation_dag.py
Automatic execution
Each day at 00.00h
dataset_sensor
Get last 'dataset_creation' execution date
Check if last execution state is success
evaluate
Read 'test_data.csv'
Call 'evaluation' function for each model
Save metric results in 'historical.csv'

Figure 22: Definition of the DAG for models evaluation.

The evaluation DAG is composed of two tasks, where the first one consists of an external sensor. Sensors in Airflow are types of tasks designed to wait for an event to occur, in this case, it waits for the last execution of the dataset generation DAG to complete successfully. Once it waits for the above DAG to be completed, the task corresponding to the evaluation of the models is executed. This is a way to ensure that the evaluation will be performed with the latest version of the data available.

The second task calls the 'evaluation' function located in the additional 'scripts' folder. This function is executed for each of the models in the repository. First, it reads the test set, separating the target variable to be used as ground truth when making predictions. Subsequently, the model prediction is performed and the predicted sales are obtained, which together with the ground truth are used to calculate the different metrics by means of their corresponding equations. Finally, a row is generated in 'historical_dataset.csv' with the results of the metrics among other information, such as the name of the model, its hyperparameters and the date on which the evaluation was carried out. This information is used to monitor the models over time, since each time they are evaluated, a new row is created with the results.

5.3.2 Training DAG

The training DAG consists of the training process of the model with the hyperparameters that the user has selected in the dashboard. Once trained, it is immediately evaluated to obtain the value of the metrics and record it in the 'historical_dataset.csv'. The diagram of the training DAG is shown below:

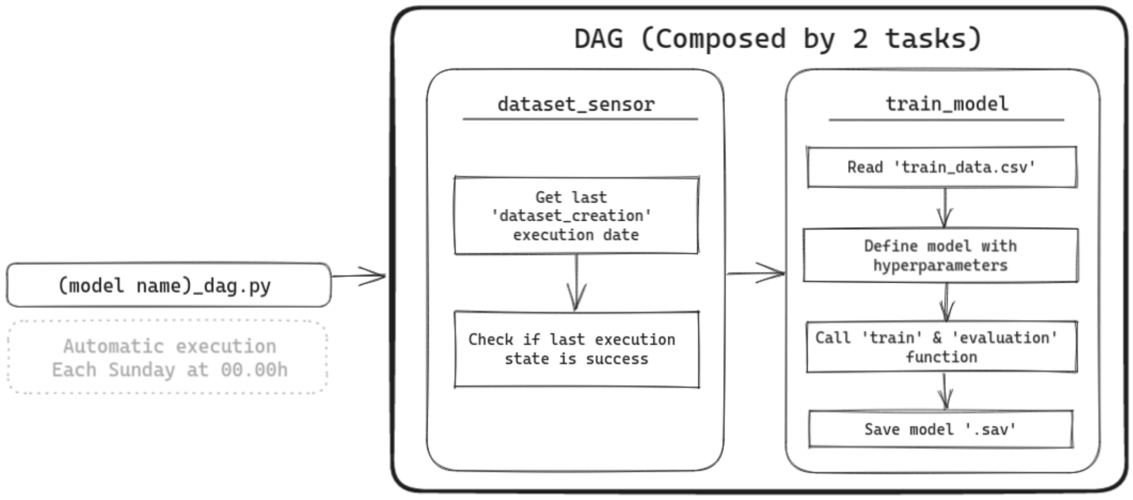

DAG (Composed by 2 tasks)
dataset_sensor
Get last 'dataset_creation' execution date
Check if last execution state is success
train_model
Read 'train_data.csv'
Define model with hyperparameters
Call 'train' & 'evaluation' function
Save model '.sav'
(model name)_dag.py
Automatic execution Each Sunday at 00.00h

Figure 23: Definition of the DAG for model training.

The training DAG has a similar composition to the evaluation DAG, where the first task consists of a sensor and the second task in this case consists of training the model. The sensor waits for the last DAG run of the dataset generation to be successfully completed, so that the model will be trained with the latest version of the data.

The model training task defines the model with the hyperparameters selected by the user. Once the model is defined, the 'train' function defined in the 'scripts' folder is called, which reads the data used for training. Then, the target variable is separated from the data and the training is executed using it as ground truth. Once the training is finished, the 'evaluation' function is called to obtain the metrics of the trained model. All this information is written in a new row of the 'historical_dataset.csv' that will include the date of the training along with the metrics obtained among other information.

It should be noted that each training DAG is specific for each model, since the model and its hyperparameters are defined according to the user's selection on the dashboard. To achieve this, a python file has not been created for each possible model since it is unfeasible, so a system of automatic generation of python files that define the DAG according to the selected model has been developed. This system will be explained in detail in the 5.4 section.

5.3.3 DAG Scheduling

The scheduling of the DAGs is the key to the automation of the MLOps framework. Each of the three DAGs presented in the previous sections is scheduled to run periodically. The DAG corresponding to model evaluation and dataset creation are scheduled to be executed every day at midnight. It is decided to run them once a day because the dataset is a time series that records each passing day with the units that have been sold of the products. In this case, the data generation DAG must be run every day to simulate the input of the new product sales of the day. On the other hand, whenever new data is added, all existing models are evaluated to have a continuous and automatic monitoring of their performance over time. In contrast, the DAGs corresponding to model training are scheduled to run once a week, specifically on Sundays at midnight. It is not considered necessary to train the models every day, since in a case where the number of models is high, this could lead to a large consumption of resources. In addition, it is possible that the difference in the performance of the models would not be significant if only one day's data had been added. For this reason, it was decided to schedule the training to run once a week, so that seven days of data would have been added. Considering also that the user can order the training of a model at any time, it is a viable option.

All DAGs are scheduled to run at 12pm, which means that at midnight on a Sunday, all three DAGs would be run at the same time. Training and evaluation can be run in parallel without any conflict. The important thing is that the data creation DAG is run before the others so that the models work with the updated data. To achieve this, the two sensors have been added to the evaluation and training DAGs as the first task of each, so that when they start running all at once, they will always wait for the new data creation to finish first. A diagram of the order of execution of the DAGs is shown below:

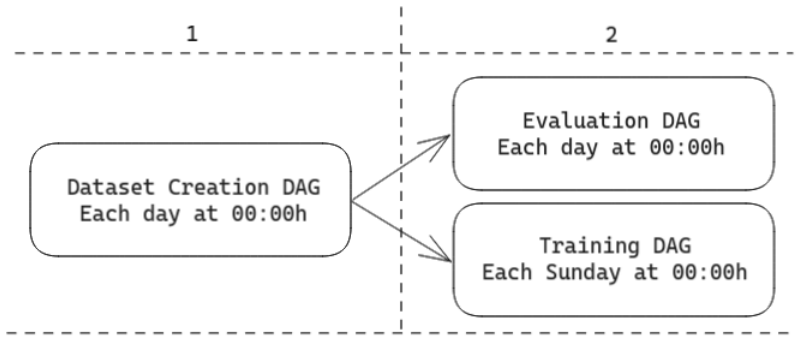


Figure 24: Order of execution and scheduling of DAGs.

5.4 Dashboard

The dashboard is the last block that makes up the designed MLOps framework. As mentioned in the design section, it is composed of four pages: Main, Managing Models, Monitoring Models and Training Models. Except for the Main page, which is used to give the user a short introduction to the framework, the rest of the pages have a series of back-end processes that are carried out in each of the available actions. These actions, such as model training or evaluation, can be manually ordered by the user through buttons, which have already been explained in section 4. Below is a diagram that summarizes the main processes that occur on each of the dashboard pages:

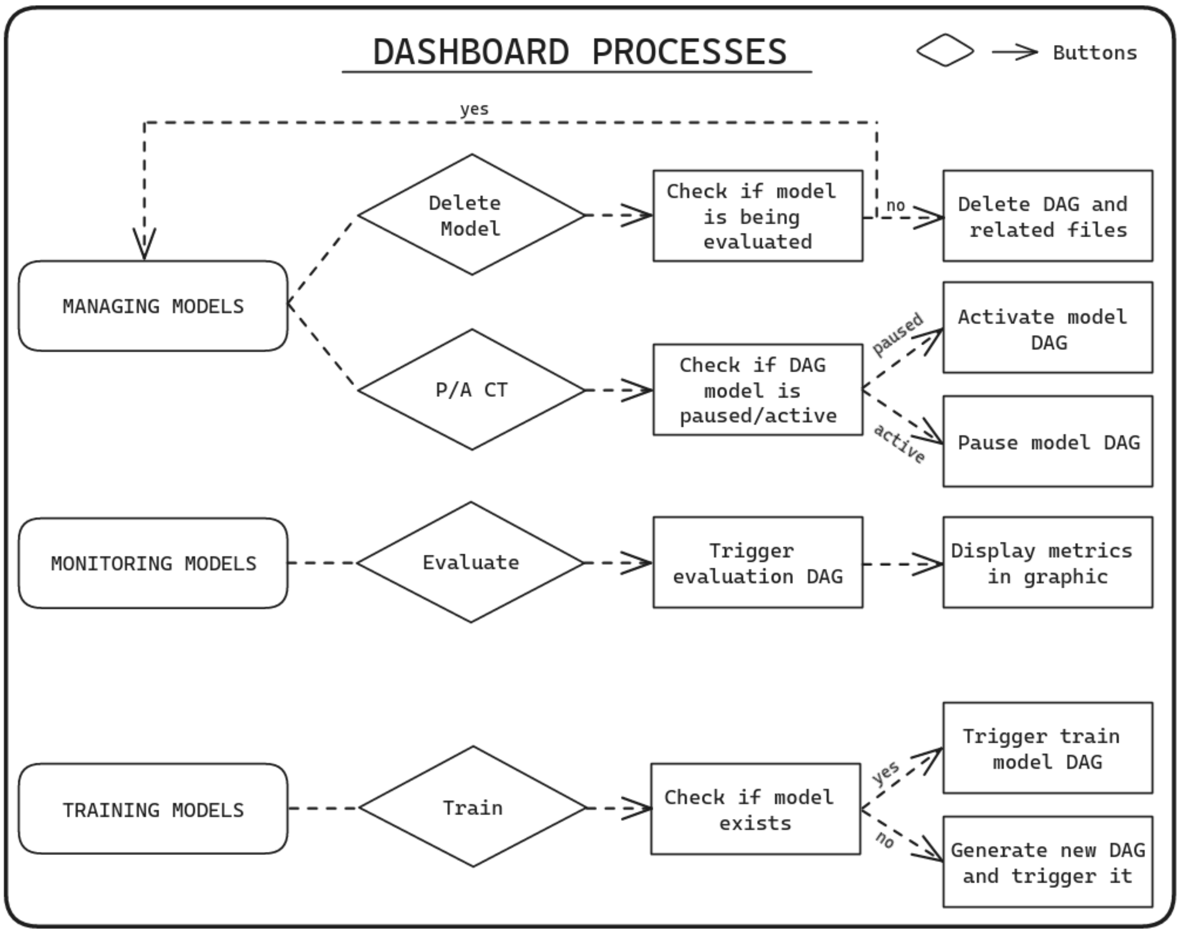


Figure 25: Main back-end processes and actions of the dashboard.

The diagram shows the fundamental processes of the dashboard, all of which are related to Airflow. Each of these processes requires communication with the orchestrator to request the trigger of a DAG or to request other types of information. The requests are made through Airflow's own REST API service. To request the trigger of a DAG or information, a series of shell scripts have been generated and programmed with the correspondent command to perform all the requests that are used. When the requests are made through shell scripts, the information returned is stored in '.json' files, which are temporary. All '.json' files and shell scripts are located in the 'coms' folder inside 'shared_volume', as they must be accessible from the dashboard.

5.4.1 Managing Models

As mentioned in section 4, the Managing Models page contains two buttons, one of which is used to delete models ('delete model') and the other to pause or activate the models' DAGs ('pause/activate CT'). When selecting a model and activating the 'delete model' button, the '.sav' file is removed from the repository alongside with the python file that defines the DAG and the rows of the 'historical_dataset.csv' that correspond to the model. In addition to deleting the python file that defines the DAG, Airflow is requested to delete the DAG from its system to ensure its removal. Once the process is completed, the user is informed by a message.

The page contains a table indicating whether the DAGs corresponding to the models are paused or active. To include this information in the table, a request is made to Airflow which returns the DAG status of each model. By selecting a model and activating the 'pause/activate CT' button, a check is made to see if the corresponding DAG is paused or not based on the information previously stored in the table. In case it is active, Airflow is requested to change its status to paused, so that the model will not be retrained every week, pausing its continuous training (CT). In case it is paused, Airflow is requested to change its status to active.

5.4.2 Monitoring Models

The Monitoring Models page has an evaluation button that can be activated by the user, as well as a graph that shows the historical performance of the existing models. When the evaluation button is pressed, Airflow is requested to trigger the evaluation DAG through the corresponding shell script. As mentioned before, this DAG performs the prediction of each of the existing models in the model repository, calculating the performance metrics and adding them to the historical graph. Once the trigger is requested, it enters a loop where the execution status is requested to Airflow every two seconds. The status of the ordered DAG execution can be success, failed, running or queued. This loop sends a message to the user on the dashboard with the execution status to keep the user informed during the process. When finished, the user can activate the refresh button to display the results on the graph.

5.4.3 Training Models

The Training Models page, like the Monitoring Models page, also has only one main button, which in this case triggers the training of the model. The model type and its hyperparameters are selected by the user before activating the training button. The user has a table with the information of the existing models to take into account when ordering the training. When the training button is activated, it checks whether the model exists or not in order to inform the user about the situation, by checking the existence of the '.sav' file in the repository and that of the python file that defines the DAG of the corresponding model. If the user has chosen to train a model that already existed in the table, he is informed that it is going to be retrained, while if it did not exist he is informed that the new model is going to be generated. It is important to check both files because an exception may occur where the DAG exists but the '.sav' file does not, which will be explained in the next section.

Each model has a unique DAG, so it has been necessary to develop a DAG generation system. This system consists of three main python files located in the 'scripts' folder in 'shared_volume' named 'dag_generation.py', 'file_creation.py' and 'dag_template.py'. After having informed the user about the existence of the template and the process to be carried out, the file 'dag_generation.py' is executed, which is prepared to run the corresponding actions depending on whether the model exists or not. In case the DAG file corresponding to the selected model already exists in the system, Airflow is requested to trigger it. The information about the trigger is saved in a temporary '.json' file that will be read back in the 'Training_Models.py' file in the dashboard section. From this temporary file, the run id of the trigger that has been ordered is read to enter a loop in which Airflow is asked for its status. Every two seconds the user is shown a message about the status of the execution until it is finished. Once finished, the new '.sav' file that has been generated in the retraining will overwrite the previous version of the model in the repository.

Back in the 'dag_generation.py' file, in case the DAG file and the '.sav' file of the model do not exist, a new DAG must be generated with the model selected by the user. First of all, the function 'create_dag' found in the 'file_creation.py' file is called. This function has as input the type of model and the hyperparameters selected by the user, and consists of copying this information in the template of the DAG to be generated. This template is defined in the file 'dag_teamplate.py', which is structured and prepared to define the training DAG, but the information about the model has to be entered. Once the information is copied, the new DAG is saved in the 'dags' folder to be detected by Airflow. This process can be slow, so Airflow is asked for the status of the DAG every two seconds, in order to know if it has been detected or not and to inform the user. This request asks for information about the DAG, which will only be obtained when it exists. When it detects that it already exists, the trigger is performed on this DAG through REST API. Finally, it returns to the 'Training_Models' file to enter a loop in which the status of the trigger is continuously requested to keep the user informed.

5.4.4 Exceptions & Error Handling

The framework is prepared to deal with a number of exceptions and errors that may occur in the execution of DAGs or in the back-end processes that are initiated when interacting with the dashboard. This section discusses how these cases are detected and how the situation is dealt with.

In all dashboard pages except 'Main', connection to Airflow is required for the execution of the different processes. A handling error case has been added in the code where the execution of processes is prevented unless Airflow is connected. If there is no connection to Airflow, the process is stopped and a message is sent to the user. To know that there is no connection to Airflow, the first request made to Airflow is used to check if there is a response. If there is no response, it means that there is no connection with Airflow, so the execution is stopped. The following is a list of situations in which this error occurs:

- On the Managing Models page, when trying to load the table that provides information on whether the DAG of the models are paused or active.
- When the 'Evaluate' button on the Monitoring Models page is activated.
- When the 'Train' button on the Training Models page is activated.

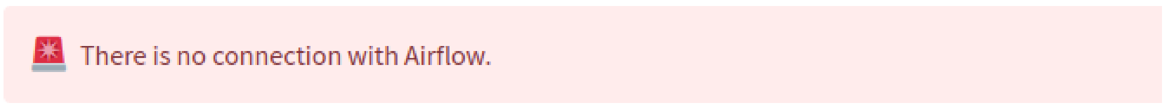
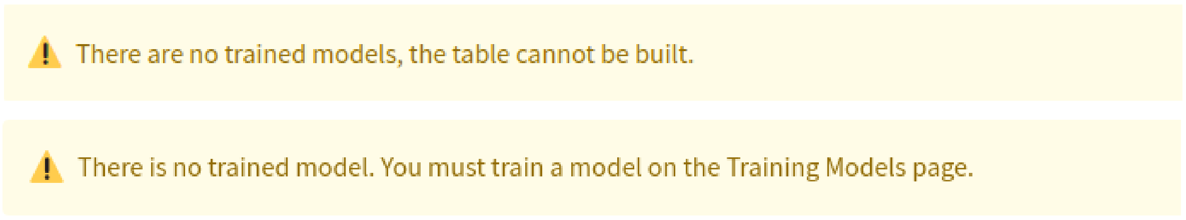

There is no connection with Airflow.

Figure 26: Airflow connection error message.

In the Managing Models and Monitoring Models pages, it is not possible to execute the available actions in case there are no models in the repository ('models' folder in 'shared_volume'). In these cases, before carrying out the process, the first thing to do is to check the number of models and stop the execution in case the number is zero. If this happens, the user is informed that there are no models by a warning message. The following is a list of situations in which this error occurs:

- In the Managing Models page, this warning will appear when trying to build the models table.

- In Monitoring Models, it will appear when the evaluation button is activated.


⚠ There are no trained models, the table cannot be built.

⚠ There is no trained model. You must train a model on the Training Models page.

Figure 27: Warning messages when there are no models.

In the Managing Models page, when selecting a model and activating the 'delete model' button it is possible that the model is involved in the execution of the evaluation DAG, since this DAG evaluates all existing models. To check it, a request is made to Airflow using the corresponding shell script, in which the list of all the executions of the evaluation DAG is requested. From this list, the last run is selected and checked to see if it is finished or in progress. In case it is in progress, the model deletion is canceled, since it could cause a conflict with the execution of the evaluation DAG. In this situation, the user is informed that he/she must request the model deletion again in a few minutes.

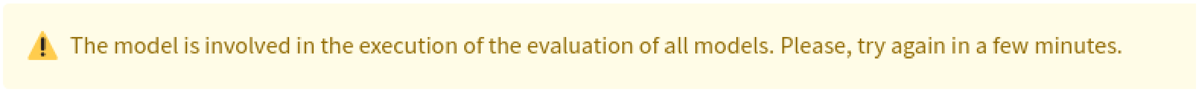
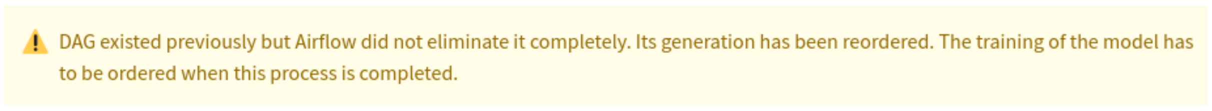

⚠ The model is involved in the execution of the evaluation of all models. Please, try again in a few minutes.

Figure 28: Warning message when the model is involved in the evaluation DAG.

The 'delete model' button on the Managing Models page can also cause conflicts when ordering training on the Training Models page of the model that has been deleted. When requesting the deletion of a model, its DAG must be deleted from Airflow, which is a time-consuming process. When deleting this model and then requesting training for it, different cases may occur for which the framework should be prepared:

- The user gives a sufficient time margin for the DAG to be removed correctly. In this case, when training the model again, the DAG will be generated following the established process.
- The user does not give a sufficient time margin for the DAG to be correctly removed. Two different situations can occur:
  - The DAG deletion has not been completed. Airflow can detect it quickly and request the training trigger.
  - The DAG removal has not been completed. Airflow can detect it immediately, but an error occurs when requesting the DAG trigger.

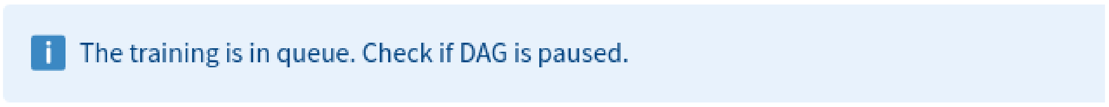
This last case is an exception that could not be replicated. It has happened repeatedly during the experiments that have been carried out with the framework throughout its development, since the deletion and training of models has been very recurrent. This has only happened when experimenting with the framework, in the case of using it correctly and consistently, it is a case that would have a very low probability of occurring. Although it is not a common case, it has been decided to treat this possible error to prevent it from causing problems. Only when the DAG that has been previously removed and it is detected immediately, i.e. on the first attempt, the execution will be stopped in order to inform the user that the procedure is being unexpected. In this case, the error that occurs when ordering the trigger is avoided, but the generation of the DAG to be trained is still in process. The user is informed with the following message:



⚠ DAG existed previously but Airflow did not eliminate it completely. Its generation has been reordered. The training of the model has to be ordered when this process is completed.

Figure 29: Warning message when training a model that has recently been deleted.

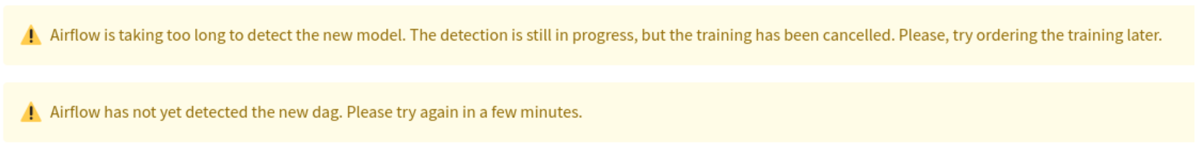
The 'pause/activate CT' button on the Managing Models page may cause an exception of which the user should be informed. By ordering a particular model to be paused, the corresponding DAG in Airflow is paused, causing the model not to be trained every certain period of time. If a model is paused and training is ordered from the Training Models page, a message will appear indicating that the status of the process is queued. The user is prompted to check if the model in question is paused, which can be easily checked in the Managing Models table. The training order remains in queue indefinitely until the DAG of the model is activated again.



The training is in queue. Check if DAG is paused.

Figure 30: Message informing that training is in queue.

The last exception is related to the generation of new DAGs when requesting the training of models that do not exist in the system. As discussed above, DAG generation is a slow process, specifically because of the time involved in detecting in Airflow the new DAG generated by means of a python file. In the 'dag_generation.py' file, a request is made to Airflow for information about the DAG in question. If no information is received, it means that the DAG has not yet been detected by Airflow. To avoid making the user wait in cases where the detection takes too long, it has been decided to request for the DAG information a maximum of 150 times, once every two seconds. When the maximum number of attempts is reached, the user is informed that the maximum waiting time has been exceeded. The DAG generation is still active but the training of the model has been canceled, so it will have to be reordered once the DAG exists in Airflow. If the training is reordered and the DAG has still not been detected, the user will be prompted to try again later.



⚠ Airflow is taking too long to detect the new model. The detection is still in progress, but the training has been cancelled. Please, try ordering the training later.

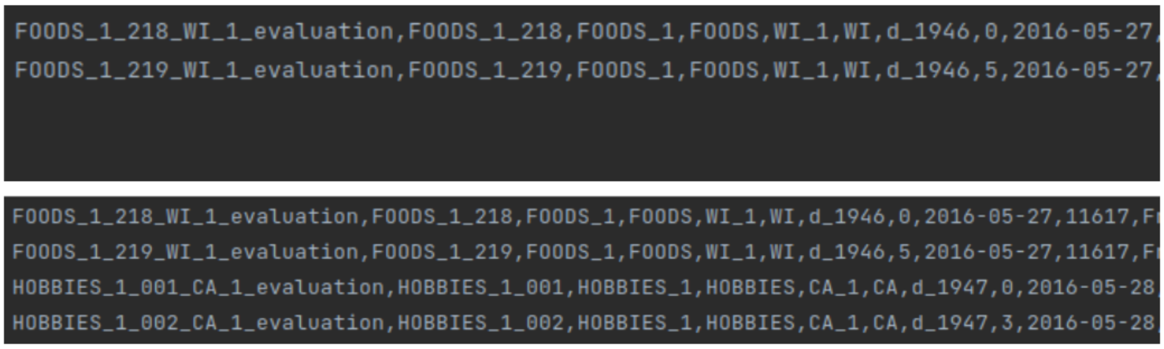
⚠ Airflow has not yet detected the new dag. Please try again in a few minutes.

Figure 31: Warning messages when waiting time is exceeded in the DAG generation.

6 Experimentation of the MLOps Framework

In the previous sections, the basics of the operation, design and development of the MLOps framework proposed in this thesis have been explained. In this section, a summary of the main operation of the framework from the user's point of view is given. The objective is to experiment with the framework, using two models and interacting with all the options offered by the dashboard.

As a starting point, we begin with a linear regression model in the repository, being the only one in operation. The model is identified by the name 'linear_T_1', where the first element refers to the type of model and the rest to the values of the hyperparameters. In this case, 'T' corresponds to the value 'True' of the hyperparameter 'fit_intercept', while '1' corresponds to the value of the hyperparameter 'n_jobs'. When starting the framework and going to the Training Models page, the user will see the screen shown in the figure 16. Once the dashboard is started, the user must start Airflow to train and evaluate models among other actions. The DAGs that have not been executed at midnight as scheduled are executed when Airflow is started. In the case of this experiment, all the DAGs have been executed as scheduled, but in order to show all the functionalities, the data generation DAG is executed again. Remember that this DAG adds a batch of data corresponding to the next day of the dataset. The following image shows the before and after of the original dataset when executing the data generation DAG:

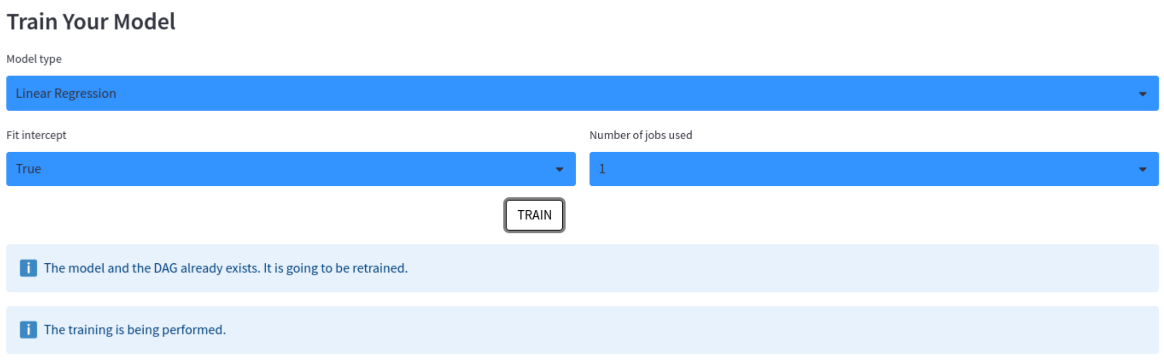


```
FOODS_1_218_WI_1_evaluation,FOODS_1_218,FOODS_1,FOODS,WI_1,WI,d_1946,0,2016-05-27
FOODS_1_219_WI_1_evaluation,FOODS_1_219,FOODS_1,FOODS,WI_1,WI,d_1946,5,2016-05-27
```

```
FOODS_1_218_WI_1_evaluation,FOODS_1_218,FOODS_1,FOODS,WI_1,WI,d_1946,0,2016-05-27,11617,F
FOODS_1_219_WI_1_evaluation,FOODS_1_219,FOODS_1,FOODS,WI_1,WI,d_1946,5,2016-05-27,11617,F
HOBBIES_1_001_CA_1_evaluation,HOBBIES_1_001,HOBBIES_1,HOBBIES,CA_1,CA,d_1947,0,2016-05-28
HOBBIES_1_002_CA_1_evaluation,HOBBIES_1_002,HOBBIES_1,HOBBIES,CA_1,CA,d_1947,3,2016-05-28
```

Figure 32: Initial data (top). Data after DAG execution (bottom).

Once the original data is modified, it is preprocessed and divided into the train set and test set. At the end of the process, the user decides that he/she wants to retrain the existing model. To do so, the user selects the model type and the corresponding hyperparameters and then presses the "Train" button.



Train Your Model

Model type: Linear Regression

Fit intercept: True

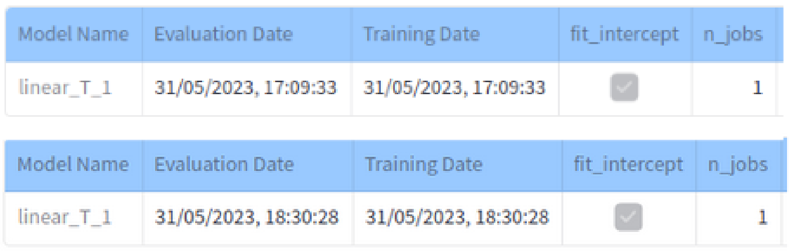
Number of jobs used: 1

The model and the DAG already exists. It is going to be retrained.

The training is being performed.

Figure 33: The retraining action is being performed.

At the end of the retraining, the '.sav' file of the model in the repository will have been overwritten by the new trained model, and the training will be recorded in a new row of the 'historical_dataset.csv'. This new row allows the user to observe the last training and evaluation date along with the resulting metrics.

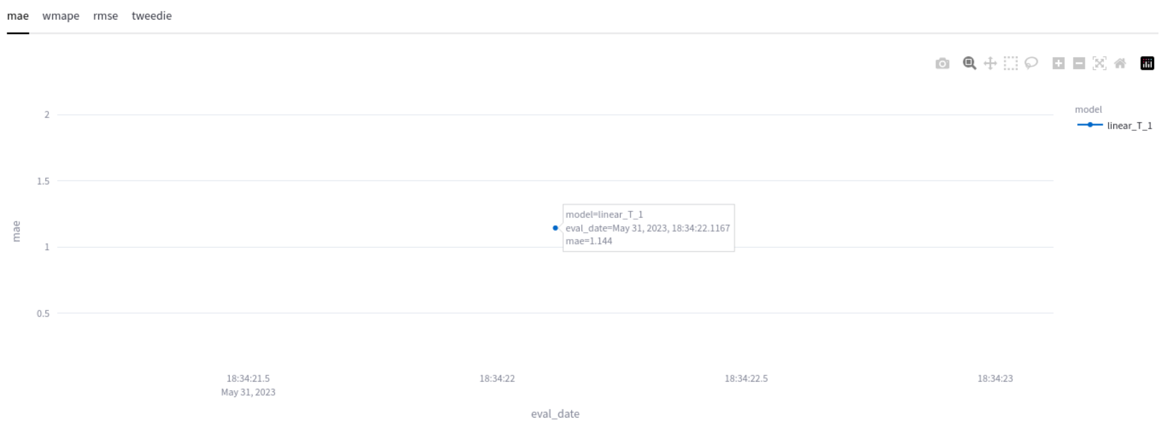


Model Name	Evaluation Date	Training Date	fit_intercept	n_jobs
linear_T_1	31/05/2023, 17:09:33	31/05/2023, 17:09:33	✓	1

Model Name	Evaluation Date	Training Date	fit_intercept	n_jobs
linear_T_1	31/05/2023, 18:30:28	31/05/2023, 18:30:28	✓	1

Figure 34: Model information before retraining (top) and after (bottom).

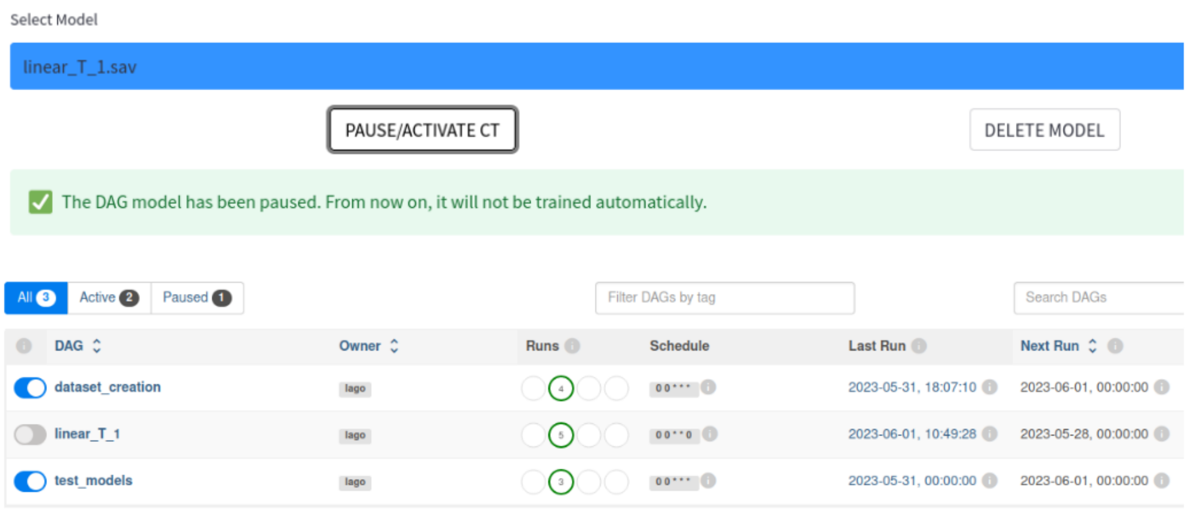
As can be seen in the image above, both the evaluation date and the training date have been updated. It is worth remembering that after ordering a training, an evaluation is also performed in order to calculate the metrics obtained by the new model instantly. The result of the metrics obtained can be seen in the graphs on the Monitoring Models page. The point added in the graph corresponding to the MAE metric is shown below:



model=linear_T_1
eval_date=May 31, 2023, 18:34:22.1167
mae=1.144

Figure 35: MAE metric added after retraining model.

Now the user decides to pause the training DAG of the model 'linear_T_1'. To do so, he/she goes to the Managing Models page, where the table of existing models and the status of its DAG (paused or active) will appear. In this case, the model is active, so the user selects it and presses the 'pause/activate CT' button to pause it. In the following image, it can be seen the dashboard message informing the user that the DAG (top) has been paused, together with the image of the DAGs in Airflow (bottom), the DAG of the model 'linear_T_1' is paused:


Select Model

linear_T_1.sav

PAUSE/ACTIVATE CT | DELETE MODEL

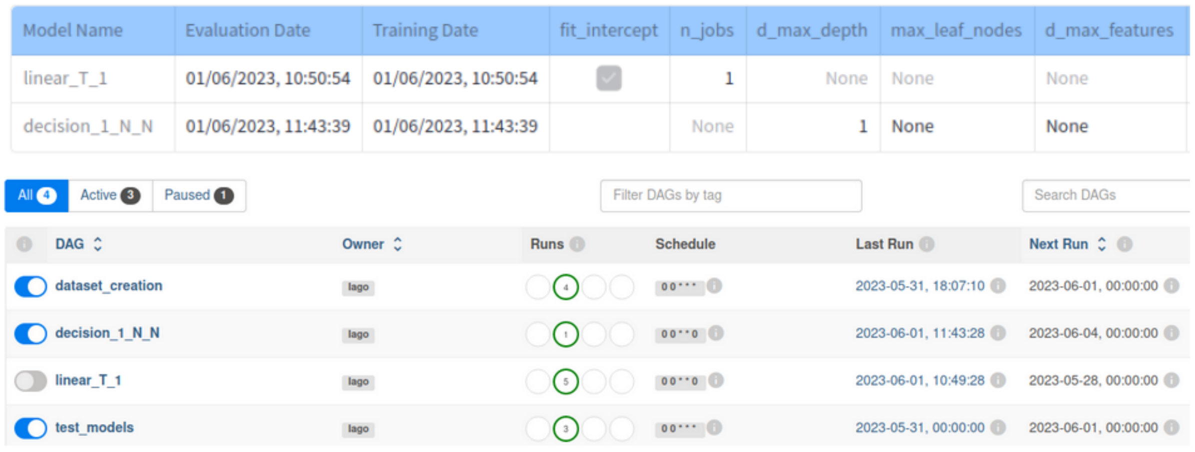
✓ The DAG model has been paused. From now on, it will not be trained automatically.

All 3 | Active 2 | Paused 1 — Filter DAGs by tag — Search DAGs

	DAG	Owner	Runs	Schedule	Last Run	Next Run
●	dataset_creation	Iago	4	0 0 * * *	2023-05-31, 18:07:10	2023-06-01, 00:00:00
○	linear_T_1	Iago	5	0 0 * * 0	2023-06-01, 10:49:28	2023-05-28, 00:00:00
●	test_models	Iago	3	0 0 * * *	2023-05-31, 00:00:00	2023-06-01, 00:00:00

Figure 36: Dashboard view of the DAG pausing process (top). Airflow view of the paused DAG (bottom).

The user then decides to generate a new decision tree model. To do so, he/she goes back to the Training Model page and selects the decision tree as the model type along with the desired hyperparameters. When the 'Train' button is pressed, the dashboard will inform the user that the model is completely new, so the Airflow detection of the newly generated DAG may take a few minutes. At the end of the process, a new DAG will appear in Airflow and a new '.sav' file will be created in the model repository located in the 'models' folder in 'shared_volume'. Below, is an image of the table on the Training Models and Airflow page, where there is evidence that the new model has been registered:


Model Name	Evaluation Date	Training Date	fit_intercept	n_jobs	d_max_depth	max_leaf_nodes	d_max_features
linear_T_1	01/06/2023, 10:50:54	01/06/2023, 10:50:54	✓	1	None	None	None
decision_1_N_N	01/06/2023, 11:43:39	01/06/2023, 11:43:39		None	1	None	None

All 4 | Active 3 | Paused 1 — Filter DAGs by tag — Search DAGs

	DAG	Owner	Runs	Schedule	Last Run	Next Run
●	dataset_creation	Iago	4	0 0 * * *	2023-05-31, 18:07:10	2023-06-01, 00:00:00
●	decision_1_N_N	Iago	1	0 0 * * 0	2023-06-01, 11:43:28	2023-06-04, 00:00:00
○	linear_T_1	Iago	5	0 0 * * 0	2023-06-01, 10:49:28	2023-05-28, 00:00:00
●	test_models	Iago	3	0 0 * * *	2023-05-31, 00:00:00	2023-06-01, 00:00:00

Figure 37: Trained models table of the dashboard (top). Airflow view of the DAGs (bottom).

Since the training of the new model, a day has passed in which a new batch of data has been added. For this reason, the user decides to order the evaluation of the two existing models in the system from the Monitoring Models page. To do so, he simply clicks on the 'Evaluation' button available on that page. The metrics obtained by each model are recorded in a row within the 'historical_dataset.csv', from which the information is extracted to build the graph with these metrics. In this way, the user will be able to observe the evolution of the model's performance. Below is the image of the graph corresponding to the MAE metric, where the new points added after completing the evaluation of the models can be observed:

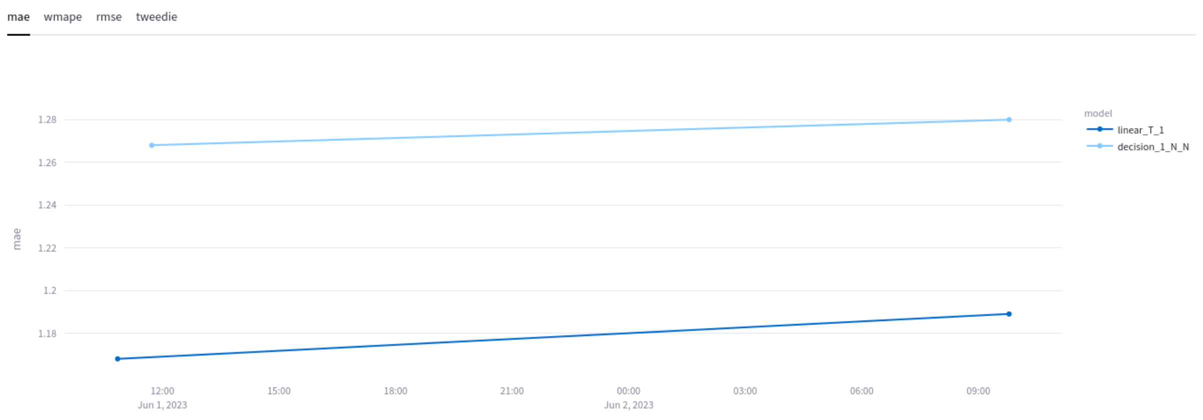


Figure 38: MAE metric graph after the model evaluation.

Due to the poor performance of the new model, the user has decided that it is not worth keeping it and that it is better to delete it. To do this, the user returns to the Managing Models page, where he/she selects the new model and presses the 'Delete Model' button. Once the deletion process is completed, you can see how the information regarding the model disappears from the dashboard, including the metrics added in the graph.

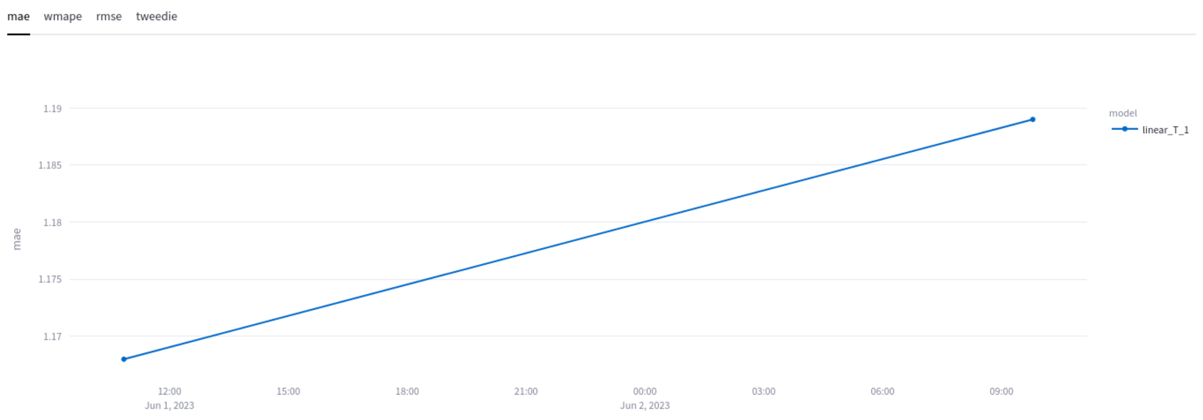
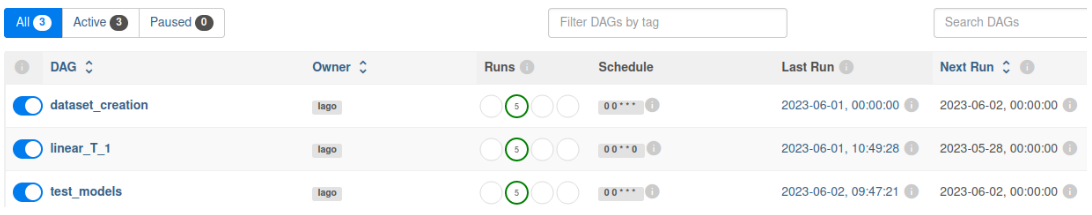


Figure 39: MAE metric graph after the model deletion.

Finally, the user decides to keep the model 'linear_T_1', so he activates again the automatic executions of its corresponding DAG. To do so, he/she selects the model and activates the 'Pause/Activate CT' button in the Managing Models page. The final list of DAGs will be as follows:



DAG	Owner	Runs	Schedule	Last Run	Next Run
dataset_creation	lago	5	0 0 * * *	2023-06-01, 00:00:00	2023-06-02, 00:00:00
linear_T_1	lago	5	0 0 * * 0	2023-06-01, 10:49:28	2023-05-28, 00:00:00
test_models	lago	5	0 0 * * *	2023-06-02, 09:47:21	2023-06-02, 00:00:00

Figure 40: Airflow DAGs final view.

7 Conclusions & Assessment

The thesis has focused on MLOps, a set of practices and tools that allow the automation and monitoring of the machine learning lifecycle. Within the field of machine learning, it is a practice that is having a considerable evolution and consideration in recent years. This is thanks to the usefulness and efficiency offered by its tools in a production environment, where time is a very valuable resource. This thesis aims to contribute to the dissemination of MLOps, studying and working in a field with a great future projection.

To meet this objective, the report has been structured in two main parts. The first part, consisting of sections 1, 2 and 3, provides a detailed introduction to the scope of MLOps, explaining what it is and how it has evolved from its origin to the present day through the publication of a series of papers. The second part, composed by sections 4, 5 and 6, explains the developed MLOps framework, which aims to collect some of the functionalities of MLOps and demonstrate its usefulness in a practical way. Together, they form a solid structure to enter this field that is becoming more and more recognized lately.

Regarding the MLOps framework developed, it is considered to have a maturity level 1 according to [10], since it includes the element of continuous training (CT) and continuous monitoring (CM). Compared to other current MLOps frameworks and platforms, the one developed in this thesis is a simpler case, since it has some of the basic MLOps functionalities. The most similar framework in terms of architecture of those that have been presented in the literature in section 2 is the one called ModelDB, which is introduced in the paper [37] published in 2016, being one of the earliest papers on MLOps. The framework they propose focuses on the management and exploration of models that have been built over time by data scientists through the use of a visual interface (front-end). Due to the evolution in the field of MLOps, the frameworks developed are becoming more and more complete, providing much more effective and efficient solutions and functionalities over time (Amazon SageMaker, S3, MLflow, etc.).

The proposed MLOps framework has the CT and CM elements, allowing the automatic training and monitoring of the models. Since we are not working directly in a production environment and due to the project scope, the continuous integration and continuous deployment (CI/CD) element has not been included. Instead, it has been decided to integrate the Managing Models page, which offers two additional actions that can be useful for the user (pause/activate TC and delete models). Furthermore, it offers the possibility for the user to order manual training and evaluation of the models, avoiding having a closed loop where all actions are scheduled to be executed in a certain time. In section 6 of the thesis, a brief practical example of the use of the functionalities offered by the proposed framework is shown. The performance of this MLOps framework demonstrates the usefulness of this set of tools in a simulated production environment, since for a data scientist to carry out monitoring by evaluating models and also training them manually in a situation where the data evolves over time takes considerable time. In addition, MLOps has the capacity to cover a considerable number of models at the same time, making it an excellent choice in such situations.

In short, the proposed MLOps framework provides a basic idea about the functioning of this set of practices. It is a development that has been carried out with the purpose of knowing, investigating and experimenting in a field that is growing significantly in the last few years. Until this recent growth, MLOps has gone unnoticed among the data scientist community, so the work in this thesis has provided me, on a personal level, with new knowledge. The publication of this thesis is intended to contribute to the dissemination of MLOps, so that more people in the community are aware of its usefulness and effectiveness.

8 Future Work

The MLOps framework developed has a number of aspects that can be improved and others that can be added as future work. The first of these is the limitation regarding the selection of models and hyperparameters. In this sense, the framework is not prepared to give total freedom to the user to build their models, unlike more advanced and professional MLOps platforms and frameworks. This is an advanced functionality but it would bring enormous value to the framework. It should be noted that if implemented, it could be considered an MLOps framework that could be used in practice and not only as a means of experimentation. For example, frameworks such as ModelDB [cite] allow the user to build a model thanks to the availability of three libraries: 'spark.ml', 'scikit-learn' and 'R'. Nowadays, there are platforms that offer freedom when building models, such as Amazon SageMaker. This platform provides an exploration mode where the user can train different models and measure their performance. It is possible to work with a wide variety of libraries, since the user can install the one he/she desires.

Another aspect of future work is regarding the model repository, data storage and model activity logs, since they are all stored in folders and '.csv' files. Nowadays, in more advanced frameworks, a cloud data storage such as the one offered by Amazon S3 or even SQL could be used, where model logs could also be stored. As future work, it is proposed to add a version control that stores the versions of the different models and data used over time. This functionality is dependent on a more efficient use of storage, since it implies a higher use of memory. Version control allows the user to consult old data or models at any time, as it can be a great source of information. Finally, adding a CI/CD element to the pipeline would increase the level of automation of the framework and make it more suitable for a production environment. The continuous deployment element can be configured to automate the deployment of the best performing model at the time. The inclusion of all these aspects would form a complete and professional MLOps framework.

A Appendix: Planning & Time Management

This appendix includes the planning and time management that has been carried out with respect to the development of the report and the MLOps framework. The master's thesis has been divided into five major tasks that have been accomplished sequentially between the months of January and June. Below is a Gantt chart showing the tasks mentioned and their corresponding dates between which they have been developed:

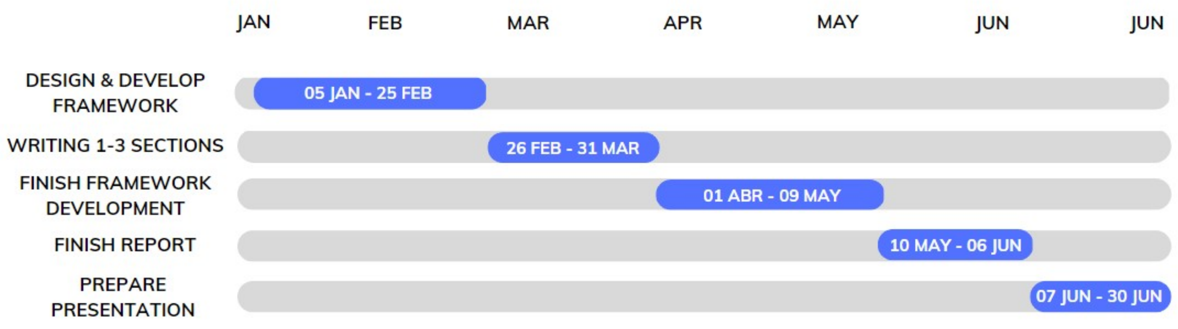


Figure 41: Master thesis planning Gantt chart.

The last task to be performed is the preparation of the thesis defense before an examination committee. The completion date of this task is approximate, as it could be set between June 26 and 30, pending official confirmation. Below is a table that includes a brief description of the tasks and the approximate total hours spent on each task:

Table 3: Tasks description and time approximation.

Task	Description	Hours
Design & Develop Framework	Design of the dashboard pages and which functionalities should include. Begins the development of the dashboard pages and the Airflow DAGs.	170 h.
Writing 1-3 Sections	Write the first sections of the report, which correspond to the theoretical part that introduces the reader to the topic of MLOps. Based on the reviewed references.	110 h
Finish Framework Development	The framework is completed and tested. The Managing Models page, exceptions of the framework and dataset generation process has been added. In addition, malfunctions in preprocessing and other processes have been fixed.	130 h
Finish Report	Write the sections corresponding to the design and development of the practical part, that is, the MLOps framework. Conclusions, future work, acknowledgements and appendix are included.	90 h.
Prepare Presentation	Prepare the slides, explanation and demonstration video to be shown to the thesis tribunal.	50 h
All Tasks	-	**550 h**

References

[1] Apache Airflow. *Scheduling & Trigger.* URL: https://airflow.apache.org/docs/apache-airflow/1.10.1/scheduler.html (visited on May 15, 2023).

[2] Apache Airflow. *Tasks.* URL: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html (visited on May 15, 2023).

[3] AWS. *¿Qué es Docker?* URL: https://aws.amazon.com/es/docker/ (visited on May 15, 2023).

[4] Nasreen Azad. "Understanding devops critical success factors and organizational practices". In: *Proceedings of the 5th International Workshop on Software-intensive Business: Towards Sustainable Software Business* (2022). DOI: 10.1145/3524614.3528627.

[5] Amitabha Banerjee et al. "Challenges and Experiences with MLOps for Performance Diagnostics in Hybrid-Cloud Enterprise Software Deployments". In: *Proceedings of the 2020 USENIX Conference on Operational Machine Learning.* 2020. URL: https://www.usenix.org/system/files/opml20_paper_banerjee.pdf.

[6] Denis M. Baylor et al. "Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform". In: *In proceedings of USENIX OpML 2019.* 2019. URL: https://www.usenix.org/conference/opml19.

[7] Ian Buchanan. *History of DevOps. How development and operations teams came together to solve dysfunction in the industry.* URL: https://www.atlassian.com/devops/what-is-devops/history-of-devops (visited on Mar. 1, 2023).

[8] Ed Burns. *What is machine learning and why is it important?* 2021. URL: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML (visited on Feb. 28, 2023).

[9] Google Cloud. *Group tasks inside DAGs.* URL: https://cloud.google.com/composer/docs/grouping-tasks-inside-dags (visited on May 16, 2023).

[10] Google Cloud. *MLOps: Continuous delivery and automation pipelines in machine learning.* URL: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=es-419#mlops_level_0_manual_process (visited on Mar. 29, 2023).

[11] Sourojit Das. *DevOps Lifecycle : Different Phases in DevOps.* 2023. URL: https://www.browserstack.com/guide/devops-lifecycle (visited on Mar. 22, 2023).

[12] Rustem Dautov, Erik Johannes Husom, and Fotis Gonidis. "Towards MLOps in Mobile Development with a Plug-in Architecture for Data Analytics". In: *2022 6th International Conference on Computer, Software and Modeling (ICCSM)* (2022). DOI: 10.1109/iccsm57214.2022.00011.

[13] Niranjan DR and Mohana. "Jenkins Pipelines: A novel approach to machine learning operations (MLOps)". In: *2022 International Conference on Edge Computing and Applications (ICECAA)* (2022). DOI: 10.1109/icecaa55415.2022.9936252.

[14] Christof Ebert et al. “DevOps”. In: *IEEE Software* 33 (2016), pp. 94–100. DOI: 10.1109/ms.2016.68.

[15] Brian Fitzgerald and Klaas-Jan Stol. “Continuous Software Engineering: A roadmap and agenda”. In: *Journal of Systems and Software* 123 (2017), pp. 176–189. DOI: 10.1016/j.jss.2015.06.063.

[16] M. Francys. *Data Scientist: ¿Qué significa desplegar nuestros modelos de ML a producción en el ámbito de los negocios?* 2021. URL: https://medium.com/datos-y-ciencia/data-scientist-que-significa-desplegar-nuestros-modelos-de-ml-a-produccion-en-el-ambito-de-los-619ea76ab47e (visited on Mar. 7, 2023).

[17] Grigori Fursin, Hervé Guillou, and Nicolas Essayan. “CodeReef: an open platform for portable MLOps, reusable automation actions and reproducible benchmarking”. In: *The Workshop on MLOps Systems at MLSys’20* (2020). URL: https://arxiv.org/abs/2001.07935.

[18] Tom Hall. *DevOps Pipeline.* URL: https://www.atlassian.com/devops/devops-tools/devops-pipeline (visited on Mar. 23, 2023).

[19] Red Hat. *What is a CI/CD pipeline?* 2022. URL: https://www.redhat.com/en/topics/devops/what-cicd-pipeline (visited on Mar. 24, 2023).

[20] Waldemar Hummer et al. “ModelOps: Cloud-based Lifecycle Management for Reliable and Trusted AI”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)* (2019). DOI: 10.1109/ic2e.2019.00025.

[21] Meenu Mary John, Helena Holmstrom Olsson, and Jan Bosch. “Towards MLOps: A Framework and Maturity Model”. In: *2021 IEEE International Conference on Cloud Engineering (IC2E)* (2021). DOI: 10.1109/seaa53835.2021.00050.

[22] Maxime Lutel. *Sales forecasting in retail: What we learned from the M5 competition.* 2021. URL: https://www.artefact.com/blog/sales-forecasting-in-retail-what-we-learned-from-the-m5-competition-published-in-medium-tech-blog/ (visited on May 10, 2023).

[23] Sasu Makinen et al. “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)* (2021). DOI: 10.1109/wain52551.2021.00024.

[24] Dusica Marijan, Arnaud Gotlieb, and Mohit Kumar Ahuja. “Challenges of Testing Machine Learning Based Systems”. In: *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)* (2019). DOI: 10.1109/aitest.2019.00010.

[25] Arthur Olga Gabriel Monteiro. *MLOps Guide.* URL: https://mlops-guide.github.io/ (visited on Mar. 27, 2023).

[26] University of Nicosia. *M5 Forecasting - Accuracy. Estimate the unit sales of Walmart retail goods.* 2020. URL: https://www.kaggle.com/competitions/m5-forecasting-accuracy/ (visited on May 11, 2023).

[27] Amit Paka, Krishna Gade, and Danny Farah. "Model Performance Management with explainable AI. Build High Performance Responsible AI". In: O'Reilly Media, Inc., 2021, pp. 2–6. URL: https://www.oreilly.com/library/view/model-performance-management/9781098108687/.

[28] Luigi Patruno. *ML in Production. Best practices for building real world machine learning systems.* 2022. URL: https://mlinproduction.com/ (visited on Mar. 7, 2023).

[29] Emmanuel Raj et al. "Edge MLOps: An Automation Framework for AIoT Applications". In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2021). DOI: 10.1109/ic2e52221.2021.00034.

[30] Danilo Sato, Arif Wider, and Cristoph Windheuser. *Continuous Delivery for Machine Learning. Automating the end-to-end lifecycle of Machine Learning applications.* 2019. URL: https://martinfowler.com/articles/cd4ml.html (visited on Mar. 13, 2023).

[31] D. Sculley et al. "Hidden Technical Debt in Machine Learning Systems". In: *Advances in Neural Information Processing Systems.* Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf.

[32] Y. Shrawanty et al. *MLOps Market Research, 2031.* 2023. URL: https://www.alliedmarketresearch.com/mlops-market-A47295s (visited on Mar. 1, 2023).

[33] Streamlit. *A faster way to build and share data apps.* URL: https://streamlit.io (visited on May 16, 2023).

[34] Rakshith Subramanya, Seppo Sierla, and Valeriy Vyatkin. "From DevOps to MLOps: Overview and Application to Electricity Market Forecasting". In: *Applied Sciences* (2022). DOI: 10.3390/app12199851.

[35] Tom Taulli. *MLOps: What You Need To Know.* 2020. URL: https://www.forbes.com/sites/tomtaulli/2020/08/01/mlops-what-you-need-to-know/?sh=1913c38f1214 (visited on Mar. 1, 2023).

[36] Matteo Testi et al. "MLOps: A Taxonomy and a Methodology". In: *IEEE Access* (2022). DOI: 10.1109/access.2022.3181730.

[37] Manasi Vartak et al. *ModelDB: A System for Machine Learning Model Management.* 2016. URL: https://cs.stanford.edu/~matei/papers/2016/hilda_modeldb.pdf (visited on Mar. 12, 2023).

[38] Naga Sanjay Vijayan. *Continuous Training of ML models. A case-study on how to keep our machine learning models relevant.* 2022. URL: https://medium.com/

@nagasanjayvijayan/continuous-training-of-ml-models-7d8acaf44dda (visited on Mar. 27, 2023).

[39] Dr. Larysa Visengeriyeva et al. *MLOps Principles. Iterative-Incremental Process in MLOps.* URL: https://ml-ops.org/content/mlops-principles (visited on Mar. 14, 2023).

[40] Matei Zaharia et al. *Accelerating the Machine Learning Lifecycle with MLflow.* 2018. URL: https://cs.stanford.edu/~matei/papers/2018/ieee_mlflow.pdf (visited on Mar. 13, 2023).