Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster en Ingenieria Informática
Master in Artificial Intelligence

Trabajo Fin de Master
Master Thesis

MLOps: Administrando el Diseño y Ciclo de Vida de los Modelos de Machine Learning

MLOps: Managing the Design and Life Circle of Machine Learning Models

Autor/Author: Alonso García Velasco

Chicago/Madrid, Julio/July - 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Trabajo Fin de Máster
Master Thesis
Máster en Ingenieria Informática
Master in Artificial Intelligence

Título: MLOps: Administrando el Diseño y Ciclo de Vida de los Modelos de Machine Learning
Tittle: MLOps: Managing the Design and Life Circle of Machine Learning Models

Julio/July - 2025

Autor / Author: Alonso García Velasco

Tutor: *Supervisor*	*Co-Tutor UPM:* *Co-supervisor:*
André Bauer	Fernando Pérez Costolla
Computer Science	Departamento de Arquitectura y Tecnología de sistemas informáticos
Illinois Institute of Technology	Universidad Politécnica de Madrid

Índice general

Índice de Figuras

Índice de Tablas

Resumen

El despliegue exitoso de soluciones de Machine Learning (ML) en entornos de producción sigue siendo un desafío significativo, ya que solo una pequeña fracción de los proyectos alcanza esta etapa. Esta tesis aborda dicha brecha proponiendo una arquitectura de MLOps (Machine Learning Operations) robusta y escalable, diseñada para optimizar todo el ciclo de vida del ML, desde el desarrollo y la experimentación hasta el despliegue, la monitorización y el reentrenamiento.

El trabajo comienza analizando las limitaciones de un sistema base previo desarrollado durante un proyecto de grado, el cual implementaba MLOps Nivel 0 utilizando herramientas de código abierto. A partir de ello, se propone e implementa una nueva arquitectura que integra MLflow, Docker, NGINX, PostgreSQL, MongoDB, Uptime Kuma y las APIs de Telegram para dar soporte a prácticas clave de MLOps como CI/CD, entrenamiento continuo, gestión de usuarios, monitorización y detección de degradación de modelos.

La plataforma permite el seguimiento centralizado de modelos, el despliegue automatizado, alertas en tiempo real sobre el rendimiento y copias de seguridad periódicas, mejorando de manera significativa la reproducibilidad, trazabilidad y mantenibilidad de los proyectos de ML. Un caso práctico demuestra todo el potencial de esta arquitectura en aplicaciones reales. Los resultados muestran una mejora medible en la fiabilidad operativa y la productividad, sentando las bases para futuras ampliaciones en áreas como escalabilidad, colaboración multiusuario y seguridad.

Abstract

The successful deployment of Machine Learning (ML) solutions in production environments remains a significant challenge, with only a small fraction of projects reaching this stage. This thesis addresses the gap by proposing a robust and scalable MLOps (Machine Learning Operations) architecture designed to streamline the entire ML lifecycle, from development and experimentation to deployment, monitoring and retraining.

The project begins by analyzing the limitations of a previous baseline system developed during an undergraduate project, which implemented MLOps Level 0 using open-source tools. Building upon this, a new architecture is proposed and implemented, integrating MLflow, Docker, NGINX, PostgreSQL, MongoDB, Uptime Kuma, and Telegram APIs to support key MLOps practices such as CI/CD, continuous training, user management, monitoring, and model degradation detection.

The platform enables centralized model tracking, automated deployment, real-time performance alerts, and periodic backups, significantly improving the reproducibility, traceability, and maintainability of ML projects. A practical case study demonstrates the full potential of this architecture in real-world applications. The results show a measurable improvement in operational reliability and productivity, setting the stage for future enhancements in areas such as scalability, multiuser collaboration, and security.

Capítulo 1

Introduction and Objectives

1.1. Introduction

1.1.1. DevOps

Software engineering emerged in the middle of the last century and, despite being a relatively new field of knowledge, it has generated methodologies and best practices for creating, using, and maintaining software since its inception.

Software development methodologies have been diverse from the beginning; among them are waterfall development, iterative development, incremental development, V-model development, spiral development, and others. In recent years, agile methodologies have become increasingly popular and follow the Agile Manifesto [1].

The agile manifesto advocates for rapid delivery of functional code to the client in order to enable successive iterations and changes to the initial requirements. This improves collaboration between developers and business stakeholders, worker motivation, fast and effective communication, maintaining a consistent work pace, focusing on technical excellence, simplicity, self-organizing teams, and frequent meetings for reflection and adjustment.

One of the most popular variants within agile methodologies is DevOps (Development Operations). It is based on the integration between development and system operations. In this model, two environments are created: preproduction or development, and production, each operated by different interconnected teams. Frequent changes can be tested in the preproduction environment before being applied and deployed to production with greater ease. This also brings other advantages, such as conducting tests in an environment that is as close to reality as possible. This allows for safer testing and supports a one-time development approach: if a piece of code works in pre-production, it should be able to move directly to production without any errors.

The DevOps process is not just about following the principles of agile development and adapting systems; it also includes restructuring development teams and providing support for end-users. Change must be supported because everything developed is intended to change at some point. The only way to achieve this result should be through an architecture that allows for change and adapts to the pace of production deployment. Another benefit of DevOps is reducing development cycles and increasing implementation speed by incorporating Continuous Integration and Continuous

Delivery/Deployment (CI/CD).

1.1.2. Artificial Intelligence evolution and history

There are some branches within computer science that do not fit directly into the DevOps methodology due to their needs, such as Machine Learning (ML) algorithms. Machine Learning is a field within Artificial Intelligence (AI). The term 'Artificial Intelligence' was coined in 1956 at the Dartmouth Conference held at Dartmouth University. In its early research years, significant advances were made both theoretically and practically. Among theoretical developments were Claude Shannon's information theory, important for analyzing variables and their relationships; or Alan Turing's computation theory, which forms the basis of computing and architectures like the Von Neumann architecture. On the practical side, the perceptron was developed, a simple AI that simulated human neurons. This era was characterized by optimism towards AI.

In the 1970s, what is known as 'AI winter' came. It received this name because advances were lacking; expectations were not met primarily due to insufficient computational power. For example, in 1975, the Cray-1[2] computer could only operate at a speed of 80MHz with just 8MB of RAM, 45 times slower than a nowadays computer with a memory three orders of magnitude smaller. The power consumption was 115 KW, three times greater than the NVIDIA A100[3]. Another limitation was the lack of large datasets that were impossible to store at the time. Additionally, one major criticism during this era was the perceptron's inability to handle non-linear classification tasks. These factors led investors and institutions to withdraw funding due to a perceived lack of progress from research teams. However, some algorithmic or imperative programming projects were actually making notable advances in AI; they certainly obtained the necessary financial resources for development. For example, NASA relied on algorithms such as trajectory planning during the Apollo 11 mission [4].

With advancements both in computational performance and paradigms (such as logic programming, specifically Prolog, Programming en LOGique [5]), it enabled the creation of rule-based expert systems towards the end of the 1980s, such as MYCIN [6]. The return to neural networks, this time with multiple layers, nonlinear activation functions, and a new way of training them in gradient descent. During these years, other projects were also launched that laid the foundations for technologies we still use today in modern AI initiatives (like autonomous vehicles), including Mercedes-Benz's 'Project Prometheus' [7].

In 1997, one of the most relevant milestones in artificial intelligence occurred with Deep Blue's victory over Garry Kasparov. During their second match, Deep Blue achieved an overall score (DeepBlue 3-1-2 Kasparov). This was particularly surprising because chess involves an astronomical number of possible moves, around 10^{120}, known as the 'Shannon number', which makes exhaustive analysis impossible. Therefore, rule-based strategies were used depending on different phases of gameplay, while storing previously played game situations and sets of openings/closed patterns.

Over the last decade, we have seen massive development in AI technologies, creating an entirely different landscape than during its first 'AI winter'. Hardware has become much more powerful and accessible; this applies both to fully managed sys-

tems internally within companies or organizations, as well as solutions using IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), or SaaS (Software-as-a-Service) provided by third-party vendors. Furthermore, the sheer volume of daily data generation versus long-term stored information has made AI implementation feasible. Companies and public entities are investing heavily in this field. In the case of Governments: United States has promised an investment of $500 Billion Dollars [8] [9] and the European Union $200 billion [10]. OpenAI and its various libraries each focusing on a distinct purpose (e.g., ChatGPT [11] generates text, Sora [12] creates images), Google did the same for their ecosystem with Gemini[13] and Veo[14], and many more companies such as DeepSeek[15], Claude[16]... In recent years, multimodal AI systems have emerged that can understand multiple data sources simultaneously using similarities between information to generate relationships or output in any form. Examples include translating dog breeds into multiple natural languages (such as English and French) and identifying them through photographs or drawings.

Figura 1.1: Multimodal AI [17]

All of this was enabled by the use of transformers [18]. Transformers were made public in 2017, and leverage the temporal understanding of RNN (Recurrent Neural Networks) without the inconvenience of sequential processing. They do this by using the attention mechanism that inspects the relationship between multiple inputs or one input by itself to include order information. This discover was first used in NLP (Natural Language Processing), video, and temporal series but was ultimately incorporated into Computer Vision and other areas that do not depend on sequential data because the extra information of order and environment was well received by models.

1.1.3. MLOps

Currently, Machine Learning developments are driven by data and have requirements that differ significantly from traditional algorithmic software development. Furthermore, depending on the source, only between 10% and 13% of ML developments reach production [19] [20]. The main reasons for this low production deployment rate are described in [21] and the methodology proposed in [22]. That article identifies several key issues: technical debt from third-party frameworks, scope creep, unexpected users, configuration problems, and environmental changes that affect the performance of ML solutions.

We are at a point where AI can deliver highly valuable outcomes for organizations. However, due to the distinct development phases involved, there is a lack of standard methodologies for ML. Fortunately, decades of experience in software engineering means that we do not have to start from scratch. One promising direction is to adapt DevOps principles to Machine Learning, resulting in MLOps.

MLOps extends DevOps by incorporating CI (Continuous Integration) and CD (Continuous Delivery), while adding CT (Continuous Training), a concept specific to ML

projects. The objective is to get models into production as quickly as possible while retaining the flexibility to modify them as often as needed, similar to how DevOps works for software.

One of the main differences is that models must be continuously tested to validate their performance over time and re-trained to improve accuracy, apply new techniques, or accommodate incoming data that were not available during the original training. Like in DevOps, the more stages of the process we can automate, the greater the productivity and reproducibility.

While ML model development is central, MLOps also involves managing:

- Configuration
- Data collection, verification, and feature extraction.
- Model testing and debugging
- Resource and metadata management
- System monitoring.

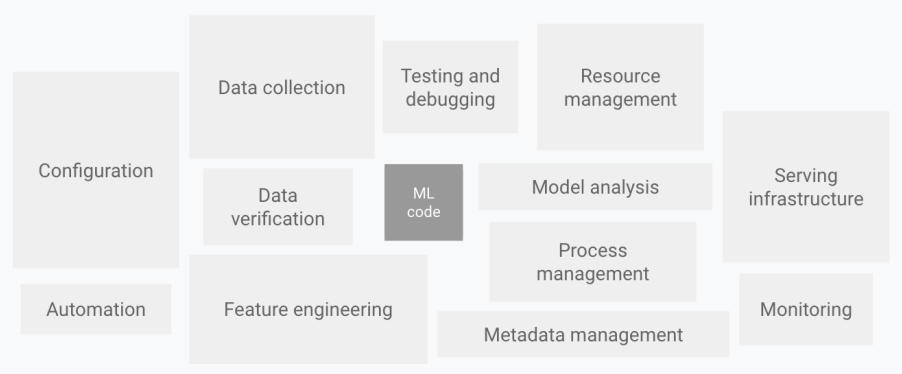

Figura 1.2: MLOps architecture [22]

As shown in Figure 1.2 [22], these components together form a full MLOps architecture.

DevOps vs MLOps

- Team Composition: ML teams often include data specialists rather than deployment engineers.
- Development Focus: Projects revolve around data relationships used to build models; code plays a secondary role and is often reused across projects.
- Testing: Model performance is typically evaluated using statistical metrics, rather than traditional unit or integration tests.
- Implementation: A model alone is not the full product; the entire process used to produce it is also part of the deliverable, usually captured in a pipeline or data flow.

- Production: Deployed models degrade faster than traditional systems, not only through typical software aging but also due to data drift. This requires continuous monitoring and periodic retraining.

Even the core principles of DevOps are adapted for ML:

- CI shifts from validating components to validating datasets and models.
- CD becomes about delivering updated data pipelines.

MLOps workflow

- Data Extraction: Obtaining data for the project to develop from the designated sources. The data sources could be multiple, for example, databases (proprietary or third party), files, API of a social network, etc. Is necessary to import said data, preferably automatic tools or ETLs (Extract Transform Load).
- Data Analysis: Getting knowledge from the data and its properties. Planning of data preparation. One of the most widely used technologies is Exploratory Data Analysis (EDA) [23]. The EDA is an study of the data and its characteristics, often aided by visualization. Helps in identifying pattern and anomalies and in validating hypothesis.
- Data Preparation: Data cleaning and separation into subsets for training, testing, and validation. Is necessary a treatment of all the data to avoid any error in them, such as making sure all the entries are in the same units or normalized. At this point, the variable selection is made.
- Model Training: Trying out different models and hyperparameter configuration in order to find the optimal configuration. Technology such as AutoML can be used to speed up the process. As an alternative, platforms such as Hugging Face [24] or GitHub [25] could be explored to find a model that suits the situation. These models would need to be fine-tuned to obtain the best results.
- Model Evaluation: The efficacy of the models is tested using the designated data subsets. The metrics are obtained from the test of this model. The last steps are repeated iteratively until the results are satisfactory.
- Model Validation: The validity of the models is tested using the validation dataset, which is a completely separated data set that is never seen by the model. This is done when the models are suitable for production.
- Model Serving. The model is used in production normally behind an API. To avoid model retraining, they are stored in a common repository and can be requested by anyone authorized to use them. This could also reduce the need for computing power as the models are trained and serve in a centralized way.
- model Monitoring: Check with regularity how good models are working, in order to change them for others in the case their capabilities decline. Data-driven decision based on the actual use metrics could be made in regards on when to retrain. Moreover, this could lead to passive dataset generation.

MLOps levels

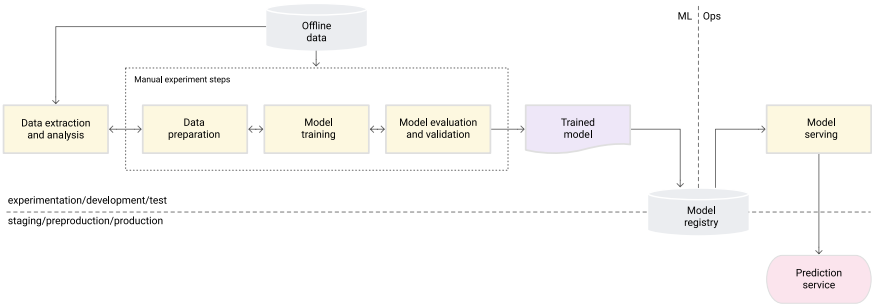

Figura 1.3: MLOps leve 0 [22]

MLOps level 0 The data processing steps are performed manually, including the transitions between the steps. There is a disconnect between the processes produced in the pipeline and the operation of the models, models must be handed off from the development team to the production team. Everything is stored in shared systems capable of retaining the necessary parameters and artifacts to execute the pipeline. From there, they can be served to production. This approach is valid if the changes to the models are few and there is no expectation of an increase in the number of models.

There is no continuous integration, delivery, or monitoring of the state of the model. It does not meet all MLOps requirements for proper development, but serves as a foundation, as there is a way to store the parameters and artifacts necessary for retraining, traceability, or deployment.

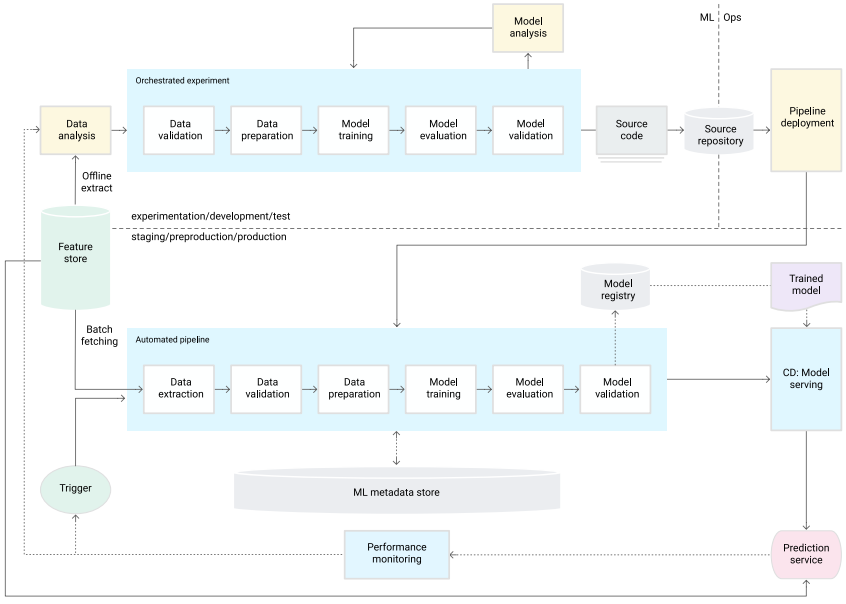

Figura 1.4: MLOps level 1 [22]

MLOps level 1 With an automated data processing pipeline, the generation of experiments becomes significantly faster. The model can be updated once in production with new data to keep it up to date. The pipelines used in both pre-production and production are the same, following DevOps principles. Continuously trained models are delivered using new, tested and validated data. Code reusability is improved because of the use of pipelines.

However, new components must be included to achieve all these advantages.

- Data Validator: Capable of automatically retraining or executing the model. It should detect anomalies or schema changes in the incoming data and inconsistencies in the values.
- Model Validator: Verifies the characteristics of the new model before it reaches production. It must perform the following steps:
  - Generate new evaluation metrics.
  - Compare them with the previous model to ensure better performance.
  - Compare metrics across different data segments. A model may perform better on certain segments, but worse overall.
  - Ensure compatibility with the infrastructure where the model will be deployed.

To increase reproducibility, the following metadata must be managed:

- Versions of the components and the pipeline.
- Start and end times of execution and its duration.
- Who ran the pipeline, the parameters used in the pipeline.
- Pointers to the pipeline steps, location of the prepared data used, anomalies during validation, statistics, and extracted vocabulary, to allow the execution to resume.
- A pointer to the last model deployed in production, enabling rollback if needed.
- Evaluation metrics generated during model evaluation for storage.
- Triggers:
  - On demand, manual
  - Programmatic triggers based on the cost of training model and the data change frequency.
  - When new data become available.
  - When model performance degrades.
  - When there is a significant change in the data.

Optionally, a feature store can be used, offering these advantages:

- Avoid duplication of feature sets that already exist or are very similar.
- Serve up-to-date features.

- Maintain consistent features between preproduction and production.

Challenges All generated pipelines must be managed, including the tests used to verify their validity and production deployment. When there are few, this can be handled manually, but as the number grows, automated management becomes necessary.

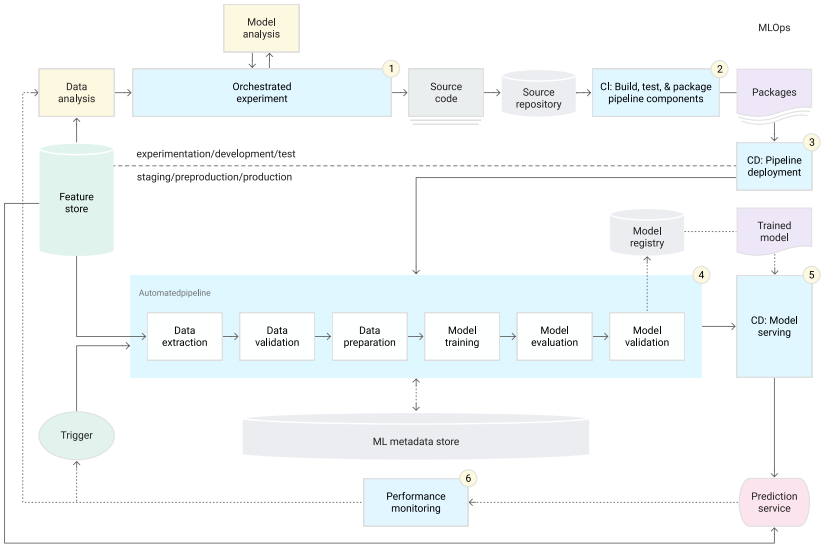

Figura 1.5: MLOps level 2 [22]

MLOps level 2, Automated CI/CD Pipeline The previous figure shows a possible Level 2 MLOps pipeline [22].

In a Level 2 environment, delivery and integration are automated, allowing the team to quickly explore new solutions, models, architectures, and hyperparameters.

It must include the following components:

- Source control.
- Services for building, testing, and deployment.
- Model registry
- Feature and ML metadata storage.
- ML pipeline orchestrator.

Pipeline workflow

- ML pipeline development and deployment, where new algorithms and models are iteratively tested.
- Continuous pipeline integration, which generates artifacts, packages, and executables for later use.

- Continuous pipeline delivery, deploying the results from the previous phase into the production environment.
- Automatic pipeline activation, either on a schedule or reactively.
- Continuous model delivery, with models served via a service.
- Service monitoring, which can act as a trigger for the process.

Continous Intregration

- Unit testing, including multiple implementations for the same step.
- Testing model capabilities and edge-case output.
- Testing that each pipeline component generates the expected artifacts.
- Testing the integration of components within the pipeline.

Continous Delivery

- Verify the compatibility of the model with the deployment infrastructure (memory, CPU, GPU, etc).
- Test the model-serving service in all parameter combinations to prevent unsupported requests.
- Test the performance of the service, for example, requests per second.
- Use automated or semi-automated test environments, depending on the methodology used to include code in the repository and test management.
- Automatically deploy to production once the proper operation is confirmed in reproduction.

1.2. Objectives

This project serves as a continuation of my TFG (Trabajo de Fin de Grado / Degree Senior project) it would treat the future lines of improvement presented in it. That is, greater automation, security, notifications, and pipelines. The objective would be part from the original and expand into a greater reach. However, this project will be an open source project, while the first one was focused on making a tool for a company (now deployed in it).

The objectives are the following.

- Analysis of the problem. General analysis of MLOps application and automatization in a company.
- Analysis of the future lines of imprement left in the TFG, tools to help improve the situation while keeping an open-source approach. Select the best approach for all of the problems presented.
- Architecture design, deployment, and configuration of the new platform with the selected methodology.

- Adapt the base-level 0 methodology implemented in the TFG to a higher MLOps level. Take advantage of all the capabilities of the new architecture.

1. Results and testing of the architecture and methodology. Apply the new technologies and methodology to a problem and follow the lifecycle.

- Conclusions Impact analysis. How does the tool work together with the methodology?
- Future lines of work: Analysis of deficiencies and future improvements. This analysis would be based on the results and conclusion discussed above.

Capítulo 2

Problem description

Currently, in the development of ML projects, only around 10 % [19] [20] actually reach production, partly due to the large amount of hidden technical debt that these types of projects accumulate through the use of third-party libraries and frameworks. One way to address this issue is to apply software development methodologies specifically tailored for ML.

To implement MLOps methodologies, we must focus on two main points. The first is the recording of all model training processes, including all the relevant parameters and information required for training. The objective is to promote the traceability and reproducibility of training runs and to select the model that delivers the best results. The second point focuses on establishing a specific development methodology for models through the use of CI/CD pipelines, combined with monitoring and life cycle management of the models. The application of MLOps can increase the value of a project by up to 30 % [26].

However, adopting these technologies poses challenges both in implementation and in the learning curve [27].

- Project Management: It is not only necessary to manage a development team, but also a multidisciplinary team that includes diverse profiles such as developers, data scientists, subject matter experts, and nontechnical personnel (management, etc.).
- Communication and Collaboration: Adopting a communication model similar to that proposed by the DevOps methodology is key.
- Everything is code:
    - Real data, the same that will be used in production, must be used to run experiments, dependencies, model retraining, and evaluation metrics.
    - Models have different life cycles than the services in which they are deployed.
    - The entire system must be reproducible through the code and its associated artifacts. Everything is defined as code: Infrastructure as Code (IaC), Configuration as Code (CaC), Pipelines as Code (PaC) to support the CI/CD structures. Development takes place in ML pipelines, and CI/CD pipelines can also be included to increase automation. Policies must be established

to allow the pipeline to reduce the likelihood of generating results from unreliable data.

- Monitoring:
  - The feature engineering and model training phases must capture all relevant metrics and experiments. Hyperparameter tuning requires careful data manipulation and needs to be captured systematically. Logging all this information enables greater reproducibility.
  - Models must be able to be monitored through an interface to analyze their performance.

Benefits of MLOps

- Productivity.
- Repeatability.
- Reliability.
- Auditability.
- Improved data and model quality.

Set of problems inherited from the TFG:

- HDFS vs. NFS/SMB: In the TFG HDFS could not implemented, and had to be changed for a simpler combination of NFS/SMB supported by a NAS.
- User management was not set.
- CI/CD was not included in the project.

New problems faced by this project:

- Security.
  - Two factor authentication.
  - Internal management of users.
- Backups.
- Pipeline.
  - Autodeployment.
  - Models from third-party repositories.
  - Saving production inputs.
  - Degradation of models
  - Autotrainning
- Control screen
  - Manage models.
  - Manage settings.

Problem description

- Recover new updates from base repository.
- Status screen.
- Notifications

Capítulo 3

Architecture Design

3.1. Analysis of the Architecture and tools proposed at the TFG

3.1.1. MLOps tool selection

In the TFG mlflow was selected as model repository. An study was carried out on all tools available at the time (January-June 2023). The following table shows the comparison made at that time:

Tool	Open/Closed source	MLOps level	Deployment location	Learning curve
Amazon SageMaker	Closed	2	Cloud	Steep
Azure Machine Learning	Closed	2	Cloud/In-premise	Steep
MLflow	Open	0 (Can be upgraded)	In-premise/Cloud	Gentle
Databricks	Closed (Automation of mlflow)	2	Cloud	Medium
Domino	Closed (Automation of mlflow)	2	Cloud/In-premise	Medium
Vertex AI	Closed	2	Cloud	Steep
KubeFlow	Open	0 (Can be upgraded)	Cloud/In-premise	Steep
H2O MLOps	Closed	2	Cloud/In-premise	Medium

Cuadro 3.1: Comparison of available tools

The TFG did not have any budget, for purchasing software licenses or for using pay-per-use systems. The two remaining platforms that support MLOps Level 0 are MLflow and Kubeflow. Both offer highly configurable open source software that can be deployed anywhere, covering MLOps Level 0, that is, centralized storage of models, artifacts, metrics, and parameters only.

MLflow provides a flexible system that can be used with any language or library. It offers the foundational components for managing various metrics, parameters, and artifacts. However, the necessary infrastructure must be implemented to store all the generated data. Its learning curve is low, requiring only basic knowledge of the MLflow Python library, just a small set of lines needs to be added to the original code.

In contrast, Kubeflow is also flexible, everything is built around containers, but it requires knowledge of containers and Kubernetes, which increases complexity.

MLflow was selected for its flexibility, ease of learning, and general usability. It comes with extensive documentation, supports the most widely used ML libraries, and offers

native support for Python, Java, and R, while also providing an REST API for other languages.

3.1.1.1. mlops features description

MLflow [28] [29] is an open source platform focused on solving MLOps challenges. It is installed as a Python library and so it can be used on any machine with Python version 3.8 or higher. It provides APIs for Python, Java, and R, as well as a REST API for languages not natively supported. Within Python, it integrates with popular ML libraries such as Scikit-learn, Keras, Gluon, XGBoost, LightGBM, Statsmodels, Spark, and PyTorch. MLflow is composed of four main modules: Tracking, Projects, Models, and Model Registry.

- Tracking: Allows for the storage of code, execution times, source code, parameters, metrics, and artifacts (any type of file produced by the training process). All elements except for the artifacts can be stored in either a filesystem or a database. Artifacts must be stored as files. Libraries are automatically captured in requirements.txt. MLflow is organized into projects; each project is formed by multiple experiments or runs. A run is a registration execution of the training process.
- Projects: Package code in a reusable and reproductible way. An API and a CLI are included to introduce these projects in the workflow. Each project is constituted by a directory or git repository that contains code. Their are called using endpoints. Dependencies are captured using python virtual environments, conda or python. Their are defined in .yaml file called MLProject.
- Models: Standard format of packaging models, each in a directory. Different types of models are allowed. The models are described in the MLmodel.yaml file. MLflow is capable of serving these models. The dependencies are automatically stored in the requirements.txt file. To use model serving, it is necessary to define entry and output schemas.
- Model registry: Is the component that centralizes the stored models, APIs, and UIs. They are used to manage the life circle of models. In older versions models could passed through different versions to control this.

Pros:

- Flexibility in development; any language and library could be used, even if they are not supported. Inside the supported ones, you get extra functionalities such as Autologing, for a streamline development.
- Functionalities are included to create ML and CI / CD pipelines.
- Open source, free and supported by an active community.
- Included as the core MLOps solution in some closed-source alternatives.

Cons:

- The tool is basically a server to obtain MLOps level 0. However, solutions are given to improve this situation.

- Currently, there is no way to calculate the precision of a model deployed.
- User authentication is still a beta feature.

3.1.2. Design, deployment and configuration presented in the TFG

3.1.2.1. Requirements

The TFG analyzed the architecture needed to deploy mlflow, formed by:

- MLflow server.
- Repository to store artifacts necessary for training and serving models.
- Database to serve metrics, hyperparameter and artifact location.

3.1.2.2. Used technologies

Docker Docker is a technology for creating containers. Unlike traditional virtual machines (VMs), virtualization is produced at the system level, rather than through a hypervisor. This means that containers are lighter and more efficient than VMs. Eliminates dependencies on the library of the system. Its use is very widespread in DevOps because it unifies the work environment to maintain pre-production and production. Docker is the most used and extended container manager, and it is open-source and has an active community. When used for deploying multiple services, such as in a microservices architecture, each module of a project runs from a separate container.

Docker provided tools used

- Docker networking - Published ports [30]: allows managing and linking virtual ports of each container to the real machine's ports.
- Docker storage - Volumes/Bind mounts [31] [32]: enables persistent storage in containers by mounting part of the real file system, managed by Docker or the operating system accordingly, into a virtual file system within the container.
- Docker compose [33]: is the native functionality to manage the deployment of multiple containers simultaneously. It allows collecting all options that would be given to the docker run command for greater replicability and ease of use. Here, you can define volumes, ports, restart options, environment variables passed to configure the system and dependencies in the startup order.

NFS NFS (Network File System) is a protocol developed by SUN to distribute file systems in heterogeneous environments. The concept it follows is to mount a branch of the remote file system on your local file system. To achieve this, the local inode points to the remote node (r-node) that you want to share. The files are stored on the NFS server.

SBM SMB (Server Message Block) is a protocol for sharing files over a network. For Windows systems, it is the standard protocol that provides this service. It is used as a substitute for NFS on Windows computers. The imported file system appears as a volume.

Virtual environments In Python, by default, packages are installed in a common repository for the entire system. This can cause compatibility problems with libraries since different projects may require different versions. The purpose of virtual environments is to encapsulate dependencies locally on the project. Another advantage is their ability to replicate development environments, providing a way to encapsulate the library dependencies used. The two most common implementations are the venv library [34] and the Conda environments [35].

They are often used together with containers to manage both system-level dependencies (containers) and library dependencies (virtual environments).

3.1.2.3. Architecture design

The architecture was designed following a bottom-up approach. MLflow could support local file storage, and step by step all of that was subtitled by network storage or a databases.

mlflow Mlflow is a Python library with many dependencies on specific versions of common Python packages. Therefore, managing it requires the use of virtual environments to avoid conflicts between packages. Mlflow does not have two separate packages to perform the functions of client and server, due to the many ways it can be deployed. This means that the client includes all the server functionality and, in some cases, performs actions that are normally handled by the server.

To ensure the independence of the operating system and implement a replicable architecture, Docker is used. To use virtual environments that manage dependencies, it is necessary for them to be activated, i.e. the Python instance used must include the appropriate libraries designed for the project. Docker containers use the root user by default, so it is advisable to use a nonroot user with the minimum required permissions to manage the container.

Mlflow is deployed as a web service, so it is necessary to specify its port and the machines it can serve. In this case, the service is intended to be accessible to all machines that can reach the server (0.0.0.0), through port 80 (standard for HTTP services). Containers expose their services on ports that are only visible inside the container. Docker provides a way to link real and virtual ports. Therefore, the basic deployment command is: *mlflow server -h 0.0.0.0 -p 80*.

Experiment, Metric, and Hyperparameter Logging Mlflow can use either a file system or a database to store experiments, metrics, and hyperparameters. A database offers better data handling in concurrent environments compared to a file system.

Through the SQLAlchemy library, Mlflow supports connecting to the following databases: MySQL, MariaDB, Oracle, MSSQL (Microsoft SQL Server), SQLite, and PostgreSQL. From these, Oracle and MSSQL can be excluded as being proprietary and paid. SQLite stores all information in a single file and is designed for minimal storage usage and efficient execution, at the expense of advanced features. Since the system may run with limited storage or processing resources, SQLite is not recommended. Of the remaining database systems: MariaDB is an open-source project created after the purchase of MySQL by Oracle (MySQL used to be open source). And PostgreSQL or Postgres is a project incited in the University of Berkeley, California. All three support

ACID properties (Atomicity, Consistency, Isolation, Durability) and rank among the most widely used databases according to DB-Engines [36]. Therefore, any of them is a good choice, but PostgreSQL has been selected.

Docker containers do not have persistent storage, so a volume must be added to recover stored data in case the container is stopped or crashes. This is especially important for databases. The database connection is made through a port that must also be mapped to a real port on the host machine to allow external access.

Artifact Logging In Mlflow, anything generated during model training and building that cannot be stored in a database is stored in a file system, which may or may not be distributed. Mlflow supports storing artifacts through the following systems or protocols: Amazon S3, Azure Blob Storage, Google Cloud Storage, FTP/SFTP, NFS, HDFS, and local file system. Amazon S3, Azure Blob Storage and Google Cloud Storage are proprietary cloud-based filesystem and paid by use service. When Mlflow is configured to store artifacts in this way, the client sends files directly to the specified server path, but this uses the client's user and an unknown group, so the server cannot access them. Using a local file system is not suitable for distributed architectures. NFS allows clients to save artifacts as if the repository were local, even though it is remote. However, the client and server must share the same path to the artifact repository, which is defined at server launch. Windows systems do not support NFS and must use SMB, which mounts file systems as volumes and then requires synchronization with the desired directory. HDFS is the best option for fault tolerance and fully distributed design, in addition to its ability to process data in parallel using MapReduce.

Due to all this, HDFS was initially chosen for its superior stability and fault tolerance. However, due to issues with setting up the HDFS server and integrating it with Mlflow, NFS/SMB had to be used instead.

3.1.2.4. Backup and Redundancy Strategy

Service Backup In the event that one of the containers crashes, it is configured to restart automatically. If the physical machine itself fails, the Docker daemon will reactivate the Docker service upon reboot and start the necessary containers.

Data Backup For simplicity during the design phase, it was proposed to use NFS and SMB, and a NAS device (Synchrony DSM) was used, which supports file sharing in both formats. This NAS system is configured to perform incremental weekly backups.

For the PostgreSQL database, the volume was mounted in such a way that it is managed directly by the file system, and an incremental weekly backup of the data has been planned.

3.2. New archirtecture and tools

3.2.1. Requirements

This new architecture and set of tools should contain the following additions.

- User loggin.
- Two factor authentication.
- Configurable backup setting.
- A way to automatically deploy models.
- A way to get models from online repositories.
- A way to control model degradation.
- A control screen.
- Be able to use HDFS or cloud storage.
- Status screen.
- A way to notify the user of backups, model degradation, and mlflow new versions.

3.2.2. Technogies used

The following technologies were discussed back the TFG:

- Docker
- Mlflow
- Virtual environments

3.2.2.1. NGINX

NGINX is an open source free HTTP web server, reverse proxy, content cache, load balancer, TCP/UDP proxy server, and mail proxy server, it also provides security in the form of user loggin. . In this case, only the reverse proxy functionality would be used. The reverse proxy is configurable to work with URLs; this means that all servides can used the / endpoint without blocking themselves. NGINX is open source and has extensive community and enterprise support.

3.2.2.2. Cron

The cron daemon is a time-based job scheduler available on Unix-like operating systems. It allows users and services to schedule scripts or commands to run at specific times or intervals.

3.2.2.3. Uptime kuma

Uptime kuma is an open source self-hosted monitoring tool with out-of-the-box support for a docker image. Unlike more complex monitoring stacks, Uptime Kuma is lightweight and easy to deploy, often used in DevOps setups for basic service-level monitoring. It provides a user-friendly dashboard and is ideal for teams or individuals looking for quick visibility into the health of the system.

3.2.2.4. MongoDB

MongoDB is a non-SQL database designed for flexibility and scalability, oriented to documents, in this case JSON. especially suitable for applications that handle large volumes of semi-structured or evolving data. In the case of this project allows for easy storage of differently structured datasets without having to change up the internal schema.

3.2.2.5. Telegram

Telegram is a messaging platform with security and automation as some of their core objectives. Telegram is known for its bots that can use processes and notifications in a DevOps/MLOps environment. Bots are easy to configure with the use of the various Telegram APIs and libraries.

3.2.3. Architecture design

3.2.3.1. Methodology

Following the steps of the TFG this project would use a bottom-up approach to the architecture design.

3.2.3.2. MLflow

The following changes to the deployment were applied:

- Customizable version.
- The server was mode to the standard mlflow port (5000) to the free port 80 for NGINX, which now redirects the traffic to mlflow.
- Services now require user and password.

3.2.3.3. Registry of experiments, metrics and hyperparameters

A postgres is still in use. A configurable backup system was added to secure the data.

The style for data backups chosen is total as is the simplest way to ensure no data is lost. To reduce the space needed to store all this data, a backup deletion schedule was also implemented.

Backups can be scheduled to run on the following schedules:

- None.
- 30 minutes.
- 1 hour.
- 12 hours.
- Daily.
- Weekly.
- Monthly.

This creates configurable environment in which is the final user the one who can adapt to the requirements of the environment they running.

The backup deletion possible schedules are the following:

- None.
- Daily.
- Weekly.
- Monthly.

Backups can not be deleted until a new one is confirmed, and backup deletion schedule must be longer than backup creation time.

3.2.3.4. Artifact registry

One of the biggest problems faced during the development of the TFG was the inclusion of HDFS as an artifact registry. At this time HDFS support is working, however, the creation of the HDFS filesystem is not covered in this project as a third party resources were used. HDFS was originally developed by Google, and they offer this kind of filesystem as service on their own cloud. Additionally, Amazon S3 is now supported. In all cases, all data are secured and backup.

3.2.3.5. Notifications module

A significant part of the problems presented in both the based mlflow repository and the TFG is that there is no notification system. Notifications are important in DevOps and MLOps project as way to control automatic pipelines or to decide that want is the best way to move forward with a certain project taking into account the most recent information.

Telegram bots and automatic email will be used. For simplicity, subscriptions to this notifications would be handle and centralized from a telegram bot. Two options will be accessible from this bot: subscribing and unsubscribing. Both would have the option to do it via telegram or email.

The notifications could be about the following themes:

- Backups.
- Mlflow updates.
- Model performance.

3.2.3.6. Update checker

MLflow is a resource with high activity in which every few minutes a new update is released. Many updates carry new features that increase the usefulness of the system. For example, from version 3.0 onwars they included a module to handle agents.

This module is in charge to extract the information from the mlflow github repository releases, save every version encounter, and in case there is a new version notify the user via the notification module.

3.2.3.7. Status checker

Uptime kuma is a health checker service that is used. It is light weight and has official docker support, which fits this project greatly. All microservices used in this project should be automatically set to be checked by uptime kuma. Moreover, Uptime kuma allows for telegram and email notifications of service loss. Unfortunately, this service does not have an official API and third-party software had to be used.

3.2.3.8. Automatization API server

As disputed before, mlflow is a level 0 MLOps tool, thus it needs the use of extra infrastructure to hold a superior MLOps level. This is where an auto-mazation API could work. Every pipeline created for the system is centralized on this server. This includes the following:

- Listing models.
- Deploying/Undeploying models.
- Consulting model information such as model signature or orignial metrics.
- Serving the deployment models (access to the mlflow model serving in a centralized way).
- Generation of data sets from actual use of the model.
- Recovery of data sets from actual use.
- Model reports, data drift and evolution.

3.2.3.9. Frontend

In order to improve the ease of use of the automatization services provided in this project, a front-end for the API was created. I do not have any experience developing front-end solution so every peace of it was created through the use of generative, in this case Github Copilot. I do not claim authorship for this work. Almost all of the API creation was made after the implementation of one or many endpoints in the API, following a pront asking the AI to create a front-end page to support the use of the API. Anything on the front-end could also be replicated through the use of scripts.

3.2.3.10. Mongo

Having said this you, there a need to store the datasets used for calling the models. For every model put into production the schema of the data used would be different from each other, thus a Non-SQL database is needed. Due to having previous experience with MongoDB, this technology was used. Its goal is to store all the data that is used to call any model on the system.

3.2.3.11. Multi API direcctions

One of the biggest problems faced at the start of this project was handling IP directions and ports. The TFG attempted to solve this, but was ultimately refused. At this time a solution involving NGINX was used. NGINX acts as a reverse proxy that diverts each call to their respective microservices. Many of these microservices are

originally developed to use the full path of API, i.e. need to have access and control '/' endpoint. This caused interference between systems; the solution found was to use subdomains. The following subdomains were used.

- $mlflow.yours\_base\_domain$ as the subdomain for the mlflow server, which should act as the primary server.
- $status.mlflow.your\_base\_domain$ as the subdomain for uptime kuma.
- $api.mlflow.your\_base\_domain$ as a subdomain for the automation server.
- $models.mlflow.your\_base\_domain$ as a shortcat to use deployed models.
- $control.mlflow.your\_base\_domain$ as a subdomain for the front-end controlling the api.

3.2.4. Archirtecture implementation

The archirtecture is based is the one presented for the TFG. At that time it only needed of two docker images, and used external storage for the artifacts, namely a NAS with NFS/SMB functionality. Now six docker machines have been used, still supported by external storage.

3.2.4.1. Mlflow

This docker machine is equal to the one presented in the TFG. The base container is built from Ubuntu version 22.04, the stable version released last year. Containers only provide the minimal and essential functionalities needed to run, so every required library must be explicitly installed. In this case, Python, wget, and libpq-dev (dependencies for using Postgres) will be required. Mlflow version is configurable, it can be set to a prefered version.

To use virtual environments, they need to be activated. To achieve this within containers, the terminal to be used must be configured beforehand, since each execution inside the Dockerfile, i.e., each RUN command, requires the environment to be automatically activated for subsequent executions. For convenience, a bash terminal will be used instead of the standard sh for containers. The –login argument will be used to force the execution of the .bashrc and .profile files, which contain the necessary information to activate the environments. For security reasons, a nonroot user should be used. This user will only have the minimum permissions necessary for the required executions and will not have a defined password.

The entrypoint.sh file that was previously defined on the host machine. They are copied inside the container, and the entrypoint.sh file is made executable.

From this point on, the non-root user is activated to complete the rest of the setup. Although conda or venv can be used for package management, using Conda is simpler when working with containers. Miniconda is downloaded. To use Conda without having to type the full path, it is added to the PATH variable. The profile file is modified to allow Conda environments to be activated, which requires initialization for Conda.

Conda repositories are updated, and the necessary environment is created for execution. Since each RUN command is executed on a different terminal instance, all li-

braries must be installed in a single execution. pip is upgraded to its latest version and mlflow, psycopg2-binary (Postgres dependencies for Python), and the library for managing HDFS from Python are installed. In addition to the TFG machine, there is a need to accept the term and conditions of using conda.

The access to the container is managed using the entrypoint.sh script. This script contains two key instructions: the first causes the execution to stop if an error is encountered; the second activates the conda environment and executes everything passed as arguments to the script.

3.2.4.2. Postgres

Both TFG and this project used just the base image of PostgreSQL. However, in this case, two new scripts were mounted inside the machine. These two scripts manage the backup system.

The first of them *make_backup.sh* handles the recompilation of the data for this task. It uses *pg_dump*, the Postgres cli tool, to complete backups. Marks the backup with the a timestamp and saves it to a share directory with the real machine.

The second script job, *restore_backup.sh* is to restore a backup to the database when needed. To do this, it lists the available backups and restores the latest one.

3.2.4.3. HDFS

In the TFG HDFS was broken down due to errors trying to store artifacts due to the fact that neither the client nor the server could log in to the filesystem. Now, to avoid the problems derived from hosting your own HDFS filesystem, it was moved to the one hosted by Google. Google offers an HDFS via Google Cloud's Dataproc.

To use Google's Dataproc you have to set up both your client (traning machine) and mflow server to gcloud and have the required permissions. This is done via the gcloud cli. This approach was discarted during the duration of the project because the mlflow server could not display correctly artifacts. Once this error is solved, the rest of the project would stay the same as mlflow can serve artifacts agnostily to the filesystem they are stored in. The rest of the project project was made using S3 instead.

3.2.4.4. S3

S3 is the blob storage present in AWS. It was the default storage used during the duration of the project. HDFS was used in general because of ease of deployment; however, a production environment could favor local HDFS over S3.

As discussed in HDFS both client and server need to have access and be Authenticated against AWS, this time there is only the need for the tokens that grant you this.

3.2.4.5. Notifications module

This docker machine handles the subscription and message of notifications. The image used is ubuntu:22:04, in addition there is python, wget, SQLite and cron. Two cron jobs are added, one handles every day at 7:00 AM if there is any new mlfow version,

while the second one handles weekly reports of the deployed models. A third Python script is used to control the subscription bot. Another script can be a call from the host machine to send messages to subscribers; this is used to handle backups. An SQLite database is used to store system users, services, and subscriptions.

The telegram bot has two main commands subscribe and unsubscribe, both are followed by the service selection and method. There are three services: backups (notifies of backups and cleanups), updates (get mlflow updates), and reports (get a weekly report of a deployed model). The method of subscription are the following telegram and email.

The script that handles the first updates uses web scraping from the GitHub releases page of mlflow [37]. Using this regex `r"releases/tag/([^\"<]*)"` (get the group after releases/tag that contains any character any number of times until a ör a <are found) the version is extracted. Versions are stored in a .csv file and compore daily. In case there is a new version, then a notification is sent to subscribers.

The other script makes a request to the automatization api for the weekly report for every deploy modelº. This report is sent to any subscriber interested in it. This also allows for the creation of the report automatically.

3.2.4.6. MongoDB

The base image of MongoDB was used with an automatic creatio of the dataset to store all the data from the model that are running.

3.2.4.7. NGINX

In this case NGINX leverages the docker internal DNS to discover all the other microservices. This allows for managin and sending multiple incoming petitions to different machines. It transforms the subdomains discuses before to their respective direction and port inside the set of containers. There is also a health checkup for the use of uptime kuma.

- `mlflow.your_base_domain` → `http://mlflow:5000`
- `status.mlflow.your_base_domain` → `http://uptime-kuma:3001`
- `api.mlflow.your_base_domain` → `http://model_deployment:8000`
- `models.mlflow.your_base_domain` → `http://model_deployment:8000`
- `control.mlflow.your_base_domain` → `http://frontend:3000`

Additionally, NGINX sets basic authentication for the services when access form the external with the use of subdomains, maintaining a passwordless access from service to service in the interior.

3.2.4.8. Model deployment API server

This machine departs from the ubuntu:24:04 image, has installed in it python, SQLite, nmap and zip. A database to hold the models that are deployed at one time containing the following data: id, model name, model version, port (the model is deployed at) and run uuid (a way to access mlflow information on the model). The

python requirements used are: fastapi, boto3(access to AWS S3), httpx, pymongo, uptime-kuma-api (a third-party api for uptime kuma), data-degradation-detector (a self-made library that makes reports on model degradation, will be discussed in the following characters) and python-multipart.

The following part will describe the inner workings of the API.

Server startup When the server starts it needs to go through various procedures that allow one to have the same environment at any time.

- Sklearn metric loadup: a map is created between metric names and the sklearn function that works them out.
- Database initialization: check if the SQLite database actually exists.
- Check the login and connection to the MongoDB database.
- Create the fastAPI app.
- Create the connection of the client to mlflow.
- Load deployed models: get a list of every deploy model from the database.
- Reload of deployed models: goes through every model in the list, check if it is running (using nmap to check if the port is open or closed) and goes through a reduced version of the deployment process.
- Set up time mapping between mlflow storage and base python typing.

Health `/health` Returns OK and status code 200, used by Uptime Kuma.

Get model list `/get_model_list` Returns a list of the names of the models in MLflow.

Get deployed models `/get_deployed_models` Returns the models that are currently running on this server.

Get model version list `/get_version_model_list` Returns a list of all versions of a defined model.

Get info of a model `/get_info/{model_name}/{version}` Returns the requirements of a deployed model given its name and version.

Get number of free ports `/get_number_free_ports` Returns the number of ports still available and not used by any model. The number of assignable ports is limited, but configurable.

Deploy `/deploy/{model_name}/{version}` The deployment methodology is the following:

- Check if the model is not already deployed.
- Get the `run_uuid` (a hash that identifies each MLflow run).

- Create a directory to hold all the information related to the model.
- Download the original data set used to train the model.
- Download the original MLflow metrics.
- Use the data set and metrics to generate the initial report.
- Get an unused port to deploy the model.
- Download the requirements of the model from MLflow.
- Add `boto3` and HDFS if they are not present.
- Use a system call to create a model directory, set up a virtual environment with `venv`, activate it, and install dependencies.
- Use the virtual environment to run: `mlflow models serve -m runs:/{run_uuid}/model -p {port} -host 0.0.0.0 -no-conda &` to deploy the model.
- Retry for up to 60 seconds using `nmap` to check if the port is open (meaning that the model is running).
- Register the model in the database.
- Register the model in Uptime Kuma.

Undeploy `/undeploy/{model_name_and_version}` This process is the inverse of deployment:

- Check if the model is deployed.
- Kill the process running the model in the background.
- Remove the model from the database.
- Remove the model from Uptime Kuma.
- Remove the directory containing the model.

Get initial report `/model/{model_name}-{version}/initial_report` Returns the files that make up the initial report for a deployed model.

Download initial report `/model/{model_name}-{version}/initial_report/download` Returns a ZIP file with the initial report files.

Get metrics `/model/{model_name}-{version}/metrics` Returns the original metrics of the model.

Update metrics `/model/{model_name}-{version}/metrics` Generates new metrics for a running model.

- Check if the model is deployed.
- Split the request body into instances (input data) and results (expected outputs).
- Call the model to get predicted outputs.

- Transform both predictions and results from JSON to NumPy arrays.
- Load the original model metrics.
- Retrieve the metric names.
- Calculate updated metrics using the mapped `sklearn` functions.
- Store the new metrics and their comparisons with timestamps.

Get new metric filenames **`/model/{model_name}-{version}/new_metrics_file_name`** Returns the filenames of all newly generated metrics for a model.

Get specific new metric **`/model/{model_name}-{version}/{filename}`** Returns a specific metric file and its comparison result.

Type mapping **`/type_mapping`** Returns the type mapping generated at startup in JSON format.

Get model signature **`/model/{model_name}-{version}/signature`** Returns a JSON object using the type mapping with the name and type of each required feature.

Get dataset **`/model/{model_name}-{version}/dataset`** Returns a dataset as a CSV file containing the data used to call the model.

- Open a connection to MongoDB.
- Check if the user specified a time range using query parameters.
- Retrieve the MongoDB dataset in that time range.
- Transform the data into a CSV file.
- Return the CSV to the user.

Generate degradation report **`/model/{model_name}-{version}/degradation_report`** Generates a degradation report for a model.

- Check if the model is deployed.
- Generate a timestamp.
- Check if the report already exists. If so, return it as a ZIP file.
- Create a directory for the new report.
- Get the original dataset used to train the model.
- Retrieve the original cluster distribution.
- Retrieve the stored data from recent predictions (weekly, monthly, yearly, overall).
- Compare the original and current metrics.
- Generate a report with all the data.

- Return a ZIP file with the report.

Proxy to model Forwards requests to deployed models. If the request is a POST to the `/invocations` endpoint (used for predictions), the input / output data are stored in MongoDB.

3.2.4.9. Docker compose

To manage all together this docker machine docker compose was used.

mlflow On build, the mlflow version can be selected, uses an.env to handle secrets, has access to AWS and HDFS. MLflow is deployed using the following command.

`mlflow server -h 0.0.0.0 -p 5000 -backend-store-uri postgresql://mlflow:$POSTGRE -default-artifact-root $DEFAULT_ARTIFACT_ROOT`. Is on the backend network using port 5000 on both real and virtual machine. Depends on postgresql.

Postgres The configuration is based on this Stackoverflow [38] thread. In addition, backup scripts are mounted on the machine. Use port 5432 and be on the back-end network.

NGINX Use port 80 on both the frontend and backend of the network. The configuration files and password management are mounted there.

Uptime kuma Uses port 3001 on both networks.

Model deployment Uses an envfile to handle secrets, is connected to mlflow, HDFS and S3 and mongoDB. Ports for models are configurable; it uses port 8000 for the server, and runs on the backend network. Depends on mlflow and mongoDB.

Frontend The port 3000 is used in both frontend and backend networks. Depends on model deployments.

MongoDB Use an envfile to store secrets, and use port 27017 on the backend.

Remote logs Uses an envfile to handle secrets, depends on postgres, mlflow, model deployment, frontend and uptime kuma. Use the backend network. Script files are mounted to allow for hot swapping of code.

3.2.4.10. Install script

This is a script that installs and configures everything necessary within the project.

- Check if docker and docker compose are installed, if not, its install it.
- Creates an envfile if there does not exist.
- Creates directories to store the backups.
- Ask and stores ebery needed environment variable:

- MLflow version.
- MLflow domain.
- Status domain.
- PostgreSQL password.
- Default artifact route (filesystem that stores artifacts).
- Start port for model deployment.
- End port for model deployment.
- AWS region.
- AWS access key.
- AWS secret access key.
- Telegram token (subscriptions bot).
- Telegram token (notifications bot).
- Telegram notifications id (Name of the bot).
- Email sender address.
- Email sneder token (tets with gmail).
- MongoDB user.
- MongoDB password.

Capítulo 4

Methodology

To properly implement the MLOps methodology, it is necessary to follow a structured approach aligned with MLOps principles.

The MLOps-compliant ML development methodology consists of the following phases:

1. Problem description.
2. Data analysis.
3. Data cleaning.
4. Experimentation and storage.
5. Model selection.
6. Model deployment.
7. Performance monitoring.
8. Retraining or replacement of the model.

Mlflow provides a repository for models, metrics, parameters, and artifacts, the new server provides an automatization pipeline. For automating data processing and treatment tasks, ETL (Extract, Transform, Load) tools are available.

4.1. Problem Description

In this phase, the problem must be described: the goals to achieve, the available data, and whether these data make achieving those goals feasible. A study of the data available during training and their sources is required, internal or external. In addition, study the techniques needed for their extraction. Acceptable error thresholds must be defined. For example, models used for medical predictions must always maintain very high quality due to their critical nature.

It is also necessary to analyze the expected duration of the model's use, whether it could be affected by data drifting, and any prior knowledge about temporal variations in data distribution. The intended usage of the model must be considered, as it may impact certain aspects; its beneficiaries, and both positive and negative implications must be evaluated.

Analyze the possibility of the model being exposed via API. Study the usual compute requirements for a model of the characteristics you want to deploy, i.e. deep learning models need to use specialized hardware (GPU) and demand much more memory. Use this information to know if you can deploy your model directly as a service or if you would need a designated machine for it.

4.2. Data Analysis

All predictive variables to be used must be understood, along with their correlation with each other and with the target variable (in the case of supervised learning), as well as their typical range of values.

This will be done through Exploratory Data Analysis (EDA), which combines statistical and visualization techniques to extract insights about data relationships. Statistical analysis helps to understand the distribution of the data, assess quality, detect missing values, and identify outliers.

One of the main tools is the correlation matrix. There are multiple methods to measure the correlation, depending on the types of variables. The NxN matrix (where N is the number of variables) shows correlation coefficients ranging from -1 to 1. The absolute value indicates the correlation strength (0: none, 1: perfect), and the sign indicates if the relationship is direct or inverse. This technique helps to easily identify the most relevant predictive variables. It is also common to include scatter plots to visualize variable correlations.

4.2.1. Data degradation detector

This library could automatically give the main statistical descriptors (mean, standard deviation, minimum, maximum, and quartiles) and automatically generate distribution plots. This could speed up the process of getting this information.

4.3. Data cleaning

For proper model training, data must meet quality standards: accurate, not duplicated, false, incorrect, or missing. Here, ETL tools can be used automatically to process data.

Duplicated data must be removed because they can distort algorithmic metrics: for example, in KNN, duplicating a point could skew predictions.

Incorrect data must be removed or corrected to avoid learning from them. This requires proper prior data analysis. There are two options: recover or discard the data. A common error is inconsistent units or scales that can be corrected. This also applies to missing-variable data.

Variables that are unique identifiers or constants should be removed, as they do not add an extrapolatable value. Variance analysis methods are used to eliminate low-variance features.

Before running experiments, ensure data validation and correctness. This phase also

includes preparing datasets for feature selection via univariate or multivariate analysis and storing them for later use.

4.4. Experimentation

Creating experiments and their runs allows you to train different models by adjusting relevant elements to obtain optimal results. Evaluation metrics, training parameters, artifacts, and generated models can be stored per experiment.

To use Mlflow, install the Python package mlflow, which includes the client and the server. Due to the many required dependencies, a virtual environment tool is recommended to prevent incompatibility issues.

As noted previously, Mlflow's strengths are flexibility and ease of use. To create experiments and runs, one only needs to modify existing scripts or notebooks. Typically, it is sufficient to import mlflow and activate autologging to automatically record the necessary data. Additional metrics, parameters, or artifacts must be logged manually.

Model training can be simplified using AutoML tools that generate multiple models quickly and tune them for precision. Mlflow does not provide this directly, but H2O AutoML does, and for free. H2O AutoML generates as many models as needed within a time limit and ranks them by selected metrics. These models can then be stored in Mlflow for comparison.

After initial AutoML model training, hyperparameters can be fine-tuned to improve results. These models are fast and effective, but additional tuning improves their performance. All trained models can be stored for future comparison.

Beyond AutoML, models can also be custom-selected by data scientists based on project-specific knowledge. Various algorithms, configurations, and variable sets are tested to find the best performing model. Feature selection or wrapper methods help avoid duplicating datasets by storing only the model path.

Mlflow's autolog can record training parameters and outputs and stores a .yaml file with the model signature, the expected input/output interface. Any other desired metadata must be manually logged.

To ensure realistic and honest metrics, evaluation must be carried out actively. The mlflow-extend library allows for storing artifacts that require preprocessing, like confusion matrices. Other important parameters, such as random seeds, can also be saved. Model summaries can also be stored as artifacts.

The new automation server needs access to the original training dataset, which must be stored as an artifact at `model_location_/data` instead of `model_location/model`.

An example of the training processes is presented in the Appendix and the attached repository [39] [40].

4.5. Model selection

In this phase, the best model from the previous step is selected. All training phase metrics and artifacts, such as confusion matrices, ROC curves, and classification reports, are used for evaluation. Adding field knowledge to this is recommended,

one option is to use cost matrices that join real the cost factor of true probables, false positives, true negatives, and false negatives with their probability of this cases occurring.

Mlflow offers tools to visualize metrics in graphs and sort experiments by different metrics, helping to identify the best model or determine if further training is necessary.

Once you are satisfied with the model you have programmed, you use mlflow UI to register the model for future use. This makes the automatization server start to see the model as ready for deployment. Each time you generate a new version of a model, you can see that reflected onto the automatization server.

4.6. Model deployment

This is where this project greatly varies from the TFG. With this setup, all models should be deployed to the centralized server described in the previous chapter.

After you have decided on a concrete model, an version to deploy you should selected for deployment on the front-end or the API. This would start a process to deploy the model and store all the information needed to create a model that is trazable and monitorazable.

All incoming data used by the models is logged to study model degradation over time. These new data may be useful for training new models or retraining existing ones.

4.6.1. Data degradation detector

This librariy produces two times of reports, the initial and the degradation. The initial's goal is to set what the model performance was at the model of deployment.

The content of this report is the following because of their:

- The statistical descriptors for each variable of the input data.
- A plot showing the distribution of each of the input data.
- The metrics the model had at this point.
- The centroid and maximum distances of a centroid to a point in a cluster from these data.
- A correlation matrix.

This report will serve as a base to compare model performance during the time is actually deployed.

4.6.2. Model Monitoring

Models must be continuously monitored to detect when performance drops below acceptable thresholds, signaling the need for retraining or replacement. Models are sensitive to trend shifts, they cannot predict patterns they have never seen.

To detect degradation of prediction capability, one must analyze newly received data, as the model may no longer recognize the patterns present in them.

Since future data distribution is unknown and unpredictable, continuous evaluation is essential.

In Figure 5.1, different types of data drift in theoretical distributions are shown. Each type of change requires a different response to ensure proper treatment. It is also necessary to keep older models available, as reverting to a previous data distribution (as in cases 2 and 4) would allow faster response times.

The problem with detecting degradation is that you rarely see a decrease in a performance metric. Models are used to predict future data and make a decision based on them. A decrease in their performance is seen after the fact, which is late. So degradation is calculated taking as a data drift, if input data is to different from expected data models a bound to not work as well. Actual performance evolution could be calculated on a model deployed, is a more definitive way to judge if the model is working well but is not always available. Some models can bypass this data drift by generalizing, and actual performance would be more accurate in them. The second report centers on giving information on data drift, but also gives out the performance evolution.

4.6.3. Data degradation detector

You could update the metrics by calling the model with both data and expected answers. This would store the new metrics and a correlation matrix for the new data.

Additionally, a complete report is made automatically each week (also on demand) is based on the data used in the previous week, month, year, and ever. This report contains the following data:

- The evolution of the data descriptors in that time frame.
- Evolution of the clusters present in the data.
- Metric evolution over time.
- Confusion matrix for each time the metric was updated.

4.7. Model Retraining or Replacement

Once the defined error tolerance has been exceeded, a new analysis must be performed to generate a new model or retrain the current one.

To do this, a condition must be defined that triggers when this threshold is reached. Once the condition is activated, the previous steps must be repeated. If the data has changed significantly, a new problem analysis may be necessary.

Data cleaning is always required; since the data has been obtained through a service, it is possible that the same information was attempted to be predicted multiple times or that predictions were made without complete data.

The scripts previously developed for cleaning the original model's data must be reused.

Once the new data is prepared, the old model should be re-trained with the new data, or a new study should be performed using AutoML or selected algorithms. All models must be registered as new runs within the original experiment. The results will then

be compared using the relevant metrics and artifacts to decide which model should be deployed as the replacement.

The model that is removed from service will be marked as archived, so it can be restored later if the data distribution shifts back to the conditions under which that model was trained. Archived models will also be included in the comparisons when selecting new models, both before and after generating new candidates.

Capítulo 5

Resultados y conclusiones

The aim of this chapter is to provide an example of a project using this methodology, testing the new modules added to handle automatization. As one of the main preoccupations of this project is data drift, a purposely drifting dataset would be created.

5.1. Problem description

The dataset selected is the Wine quality available in multiple places such as Kaggle, UCI, sklearn datasets,... is a simple and common dataset use for demonstrating purposes.

In this case to force data drift, I visualized the base target variable distribution.

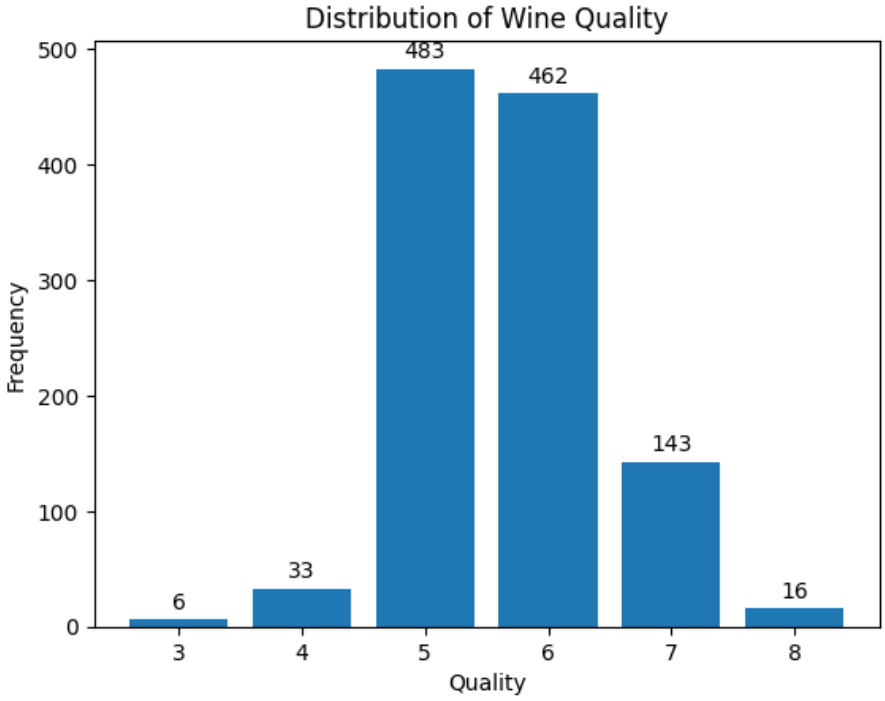

Figura 5.1: Base distribution of wine quality

As you could see in figure 5.1 the distribution could be split into two halves, bad and good wine. Bad is 5 or under and good is 6 or above.

Both classes are well balanced and you could say any of them would set a trend for data drift. For that reason, I created two datasets that are very skewed 75 %-25 % bad and good and vice versa.

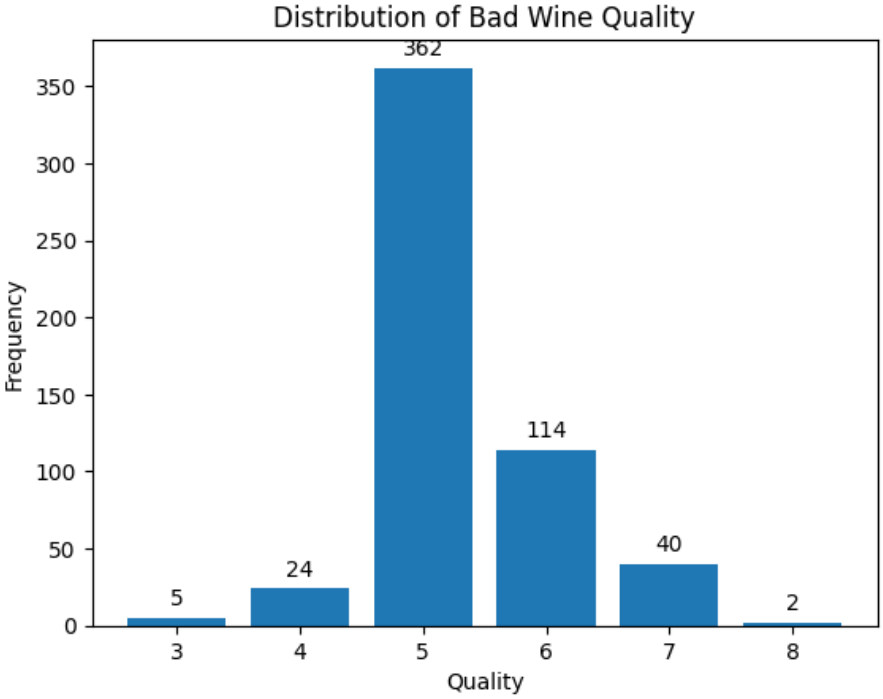

Figura 5.2: Distribution of skewed bad wine

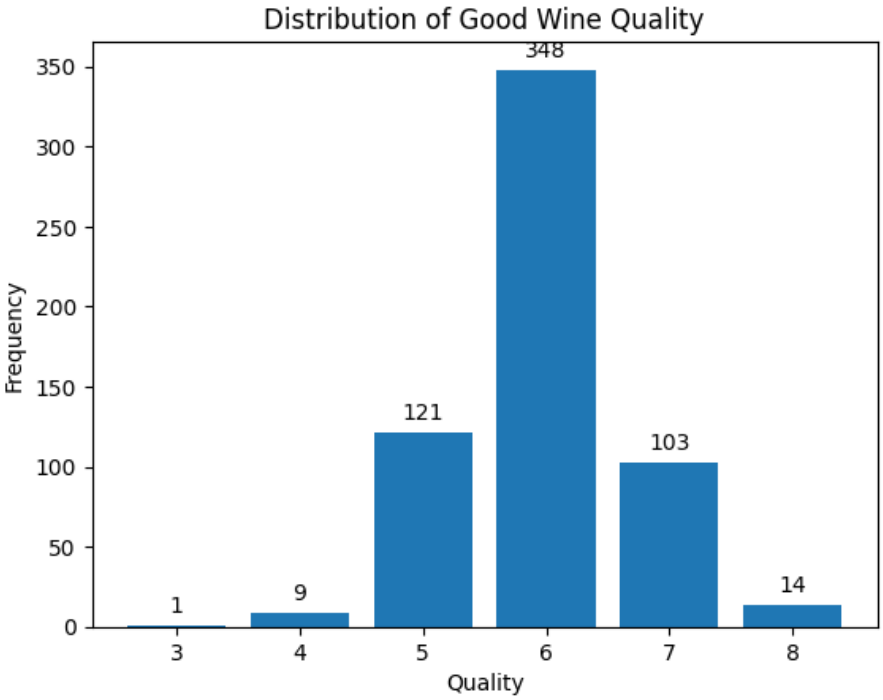

Figura 5.3: Distribution of skewed good wine

Images 5.3 and 5.4 show the skew distribution. Moreover, as a control group, I created an SMOTEd (oversampling) dataset to test both models at the end, with new

artificial data. Taking into account oversampling could lead to bad results, as those data are not real.

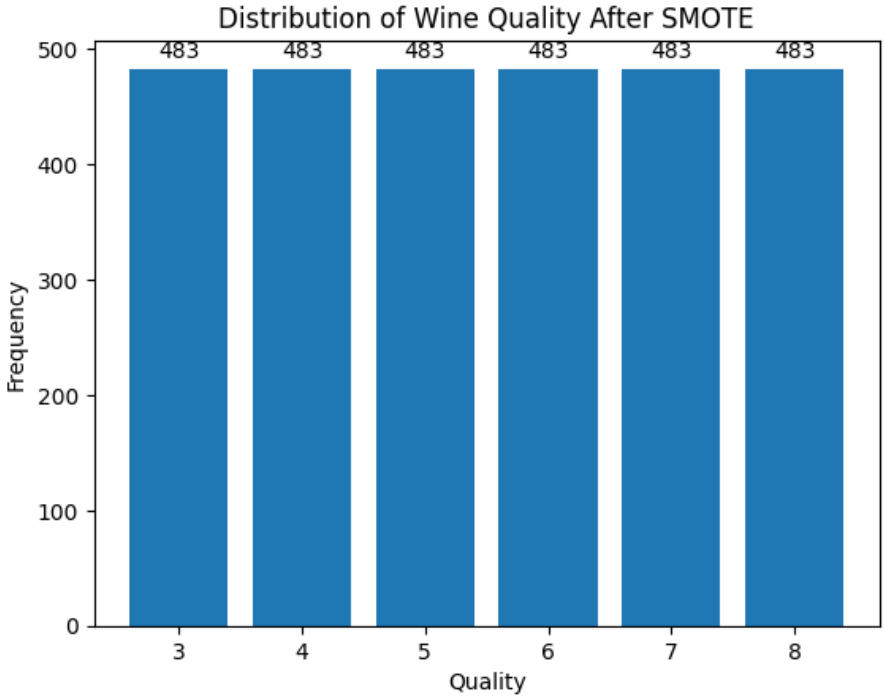

Figura 5.4: Distribution of SMOTEd wine

Figure 5.5 shows over-sampled data.

5.2. Data analysis

The first thing to do in this type of problem is to understand the data:

- **fixed acidity**: Concentration of non-volatile acids (mainly tartaric and malic) in wine, measured in g/dm^3.
- **volatile acidity**: The amount of volatile acids (mainly acetic acid), which can cause an unpleasant taste of vinegar, measured in g / dm^3.
- **citric acid**: Concentration of citric acid, a weak organic acid found naturally in citrus fruits that adds freshness and flavor, measured in g/dm^3.
- **residual sugar**: The amount of sugar remaining after fermentation, measured in g/dm^3. Low values indicate dry wine.
- **chlorides**: Salt concentration (NaCl), representing the salinity of the wine, measured in g/dm^3.
- **free sulfur dioxide**: The amount of SO_2 not bound to other compounds, acting as an antimicrobial and antioxidant, measured in mg/dm^3.
- **total sulfur dioxide**: Total SO_2 content (free and bound), measured in mg/dm^3. High levels can affect the stability and taste of the wine.

- **density**: Density of the wine, measured in g/cm^3.

- **pH**: Measure of the acidity or basicity of the wine. Lower values mean more acidic.

- **sulfates**: The concentration of potassium sulfate, which contributes to the preservation and taste of wine, measured in g / dm^3.

- **alcohol**: Alcohol content of the wine, expressed as a percentage by volume (% vol).

- **quality**: Wine quality score based on sensory data (integer score typically between 0 and 10). This is the target variable.

- **Id**: Unique identifier for each wine sample. It must be discarded.

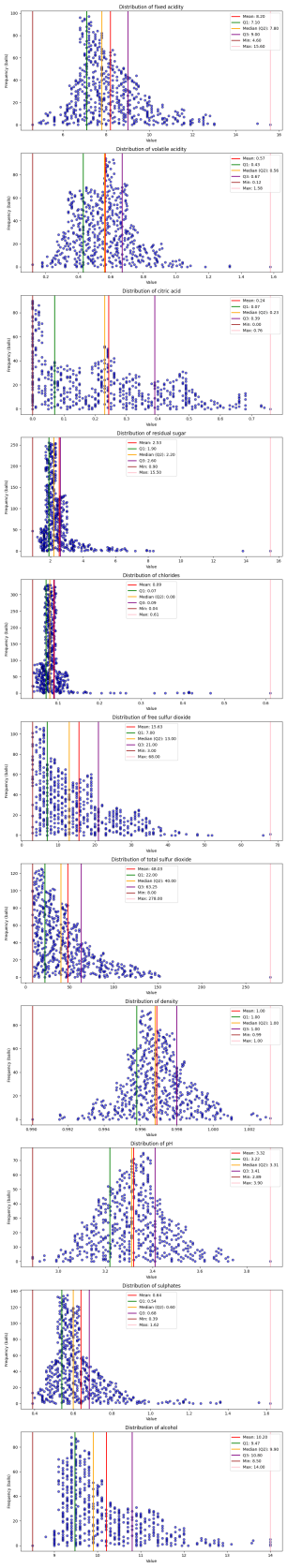

Figura 5.5: Variable distribution of all predicting metrics

In Figure 5.6 you can see that most of the data are skewed to the left of the graphic, which is not good for results.

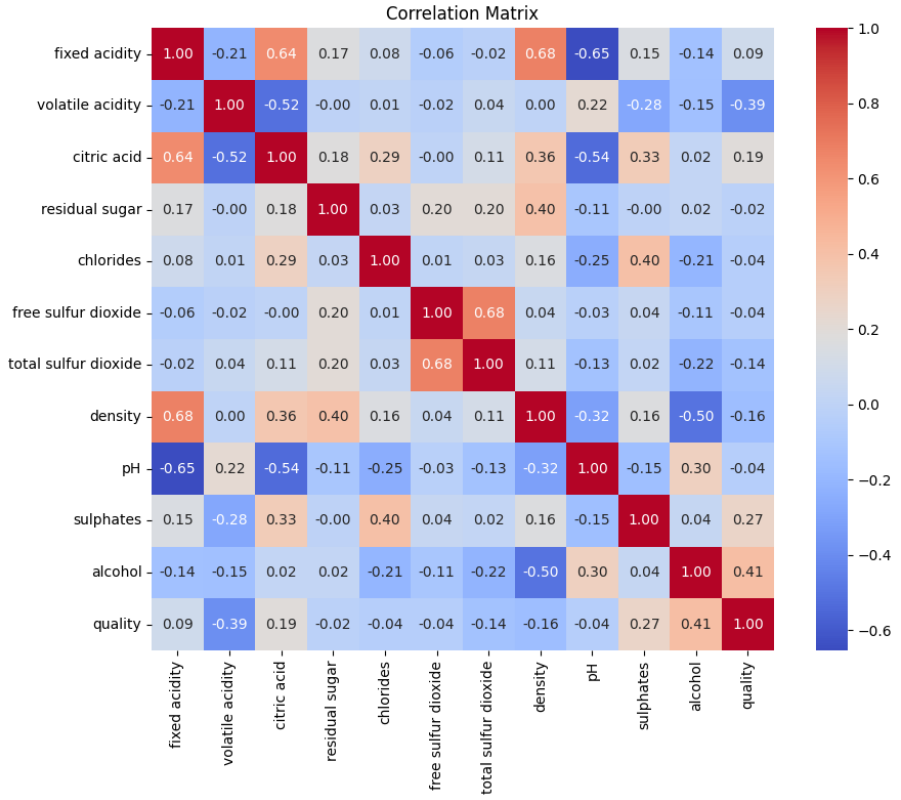

Figura 5.6: Correlation matrix

In Figure 5.7 you can appreciate that the variables related the most to wine quality are alcohol, volatile acid, followed by sulfates, and citric acid. Fixed acidity, volatile acidity, citric acid, and pH are strongly correlated. This makes sense as acids are measured in pH. The sulfates are related by themselves. And density and alcohol are also strongly related, as a higher proportion of alcohol could reduce the density.

5.3. Data cleaning

This dataset is widely used because it does not require data cleaning and could be used as is. A point could be made that you could remove the less corrected variables, but for the sake of simplicity of demonstrating the data degradation and how this system could be fixed in the long run. The only thing done with the model was the removal of the Id column.

5.4. Experimentation

I opted for H2O AutoML to train the data following the next steps.

- Start the H2O server.

- Load environment variables such as the experiment name or the loggin to mlflow.
- Centralized the metrics into a function, in this case precision, recall, f1-score and a confusion matrix.
- Splitting data into training and testing.
- Upload the data to H2O.
- Create a dataset as input example for mlflow.
- AutoML: train for a defined amount of time multiple models.
- Get the top ten models.
- For each model:
  - Calculate the metrics.
  - Upload the metrics to mlflow.
  - Upload the dataset to mlflow.
  - Upload the model to mlflow.

5.5. Model selection

Using the mlflow UI the selection of the best possible model.

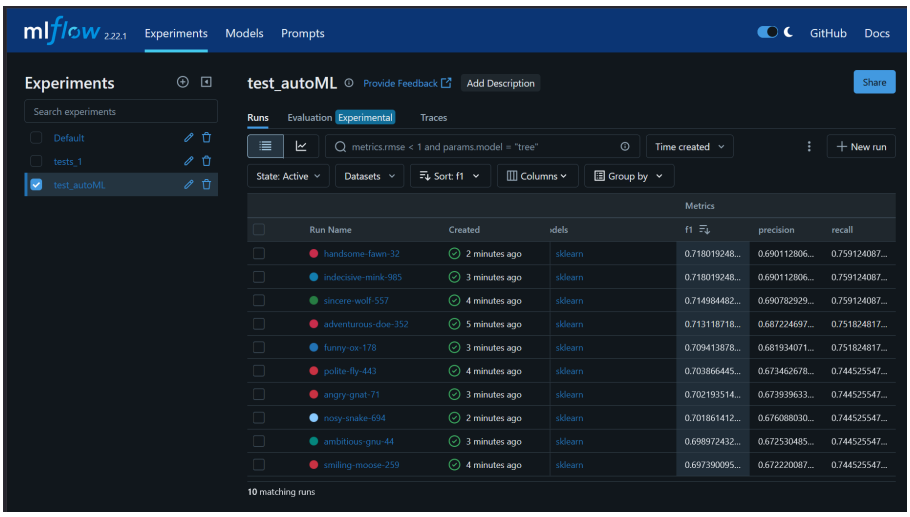

Figura 5.7: Model selecton in mlflow UI

Figure 5.8 shows that the best model taking into account f1 score is masked-skink-910, it loses one percent precision against other models, but its recall is remarkably superior. The confusion matrix apart from being biased towards the more populated classes is one of the best for this set fo models.

Taking all that information into account, this model would be used in production.

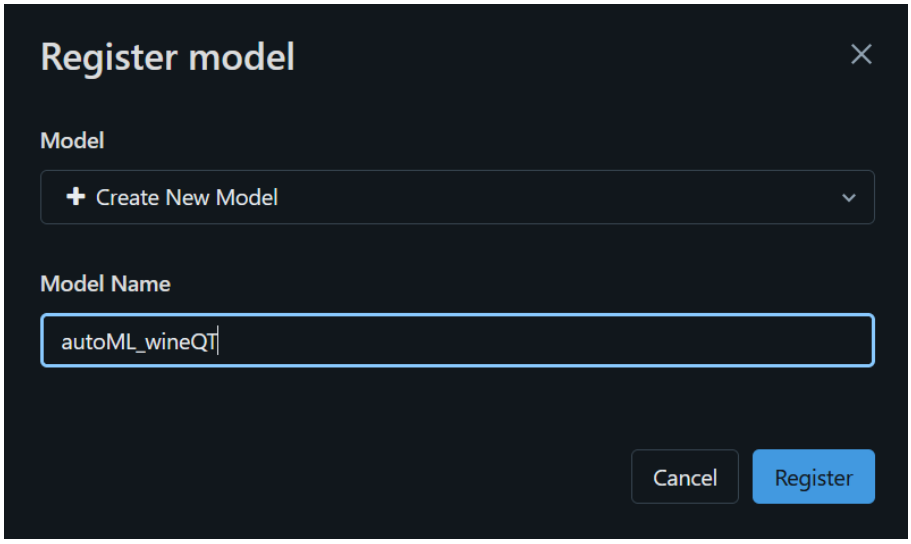

Figura 5.8: Model registration in mlflow UI

Figure 5.9 shows the model as a register.

Now by going to the automatization server frontend the model could be deployed.

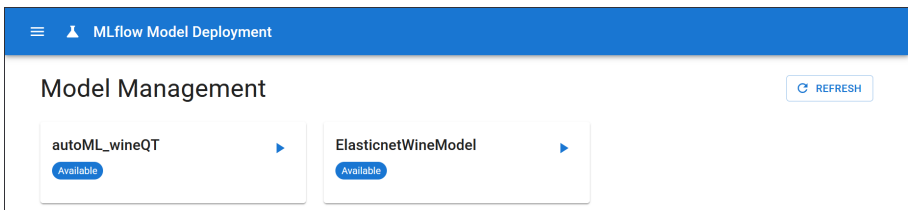

Figura 5.9: Model deploy on the automatization server frontend

And the deployment is complete by just completing the questionare with the version and number of output classes (for the degradation supervision). This step would take a few minutes to complete.

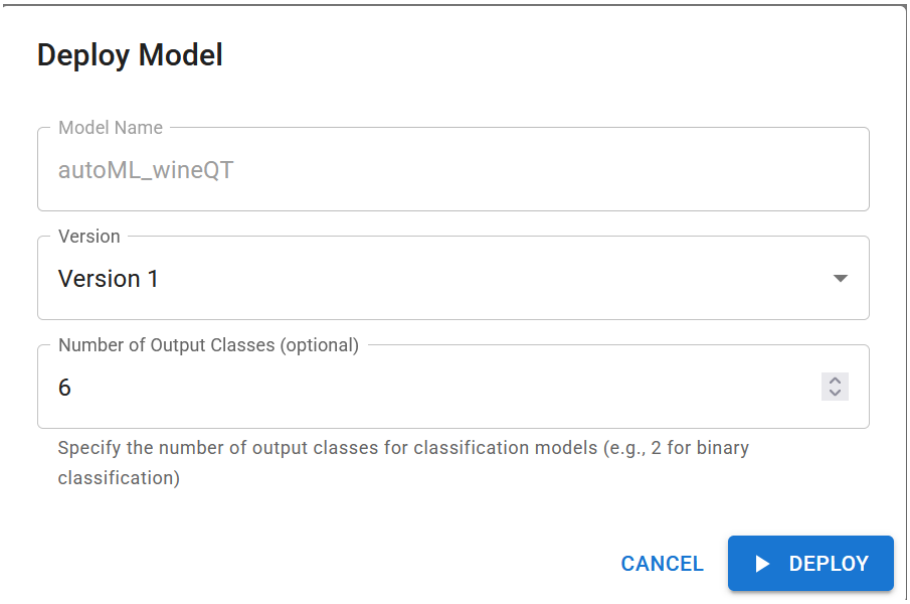

Figura 5.10: Model deploy on the automatization server frontend survey

You can check model metrics on the a frontend page, in which you could also test how the model will handle the new data that is now available.

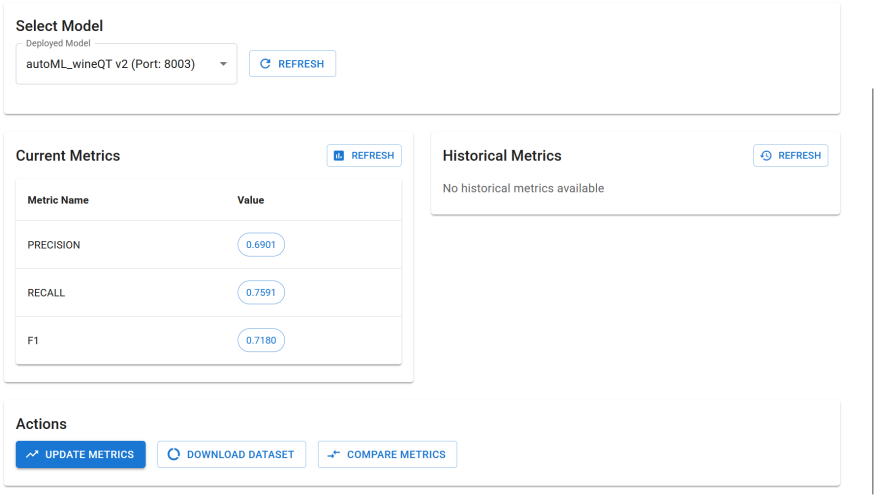

Figura 5.11: Visualization of new metrics

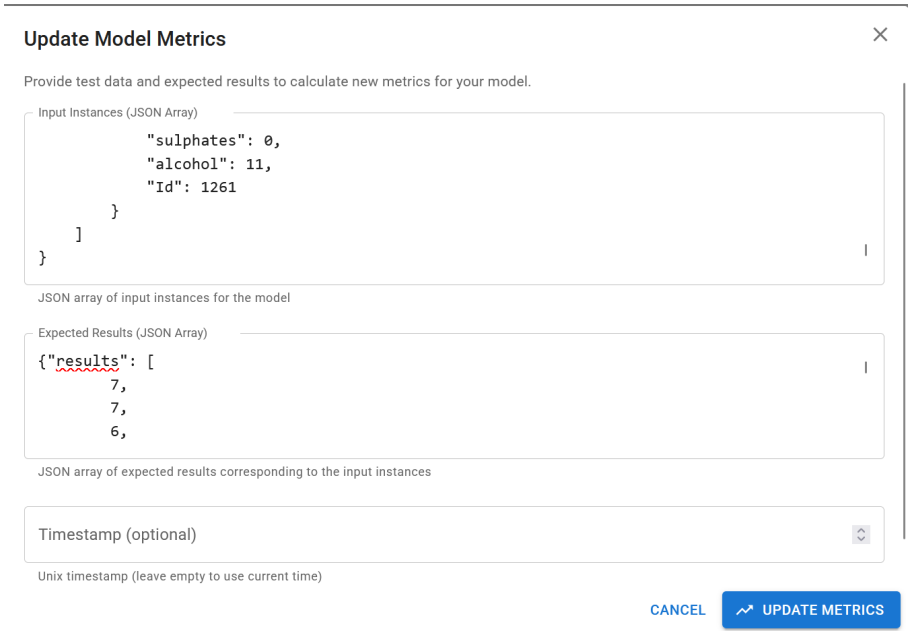

Figura 5.12: Upadate survey for metrics

After that we could check the new metric on the same page.

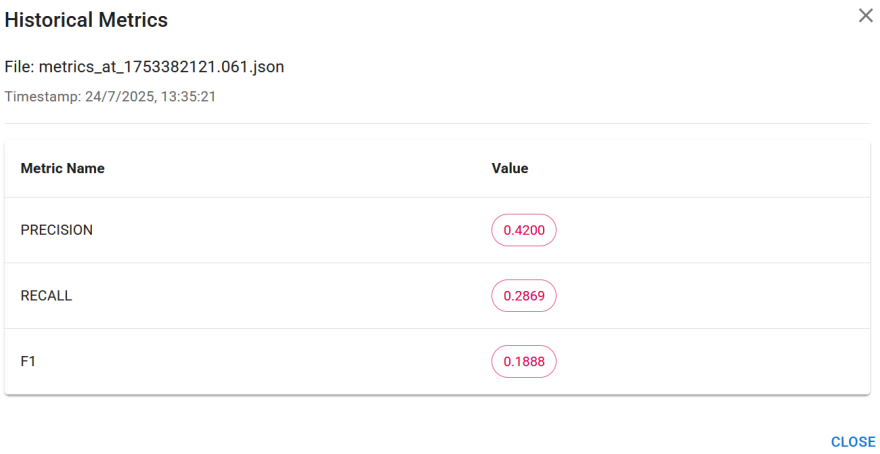

Figura 5.13: Data drift view in metrics

The degradation of the model is huge; now we could also check the full report on the automatization server. In the degradation report, one of the most descriptive changes is the clustering changes report. The centroids have not changed either the radius nor the radius, but all of the percentages of data covered by each of them are changed by more than 15 %-30 %. Probably because of the inclusion of the new data.

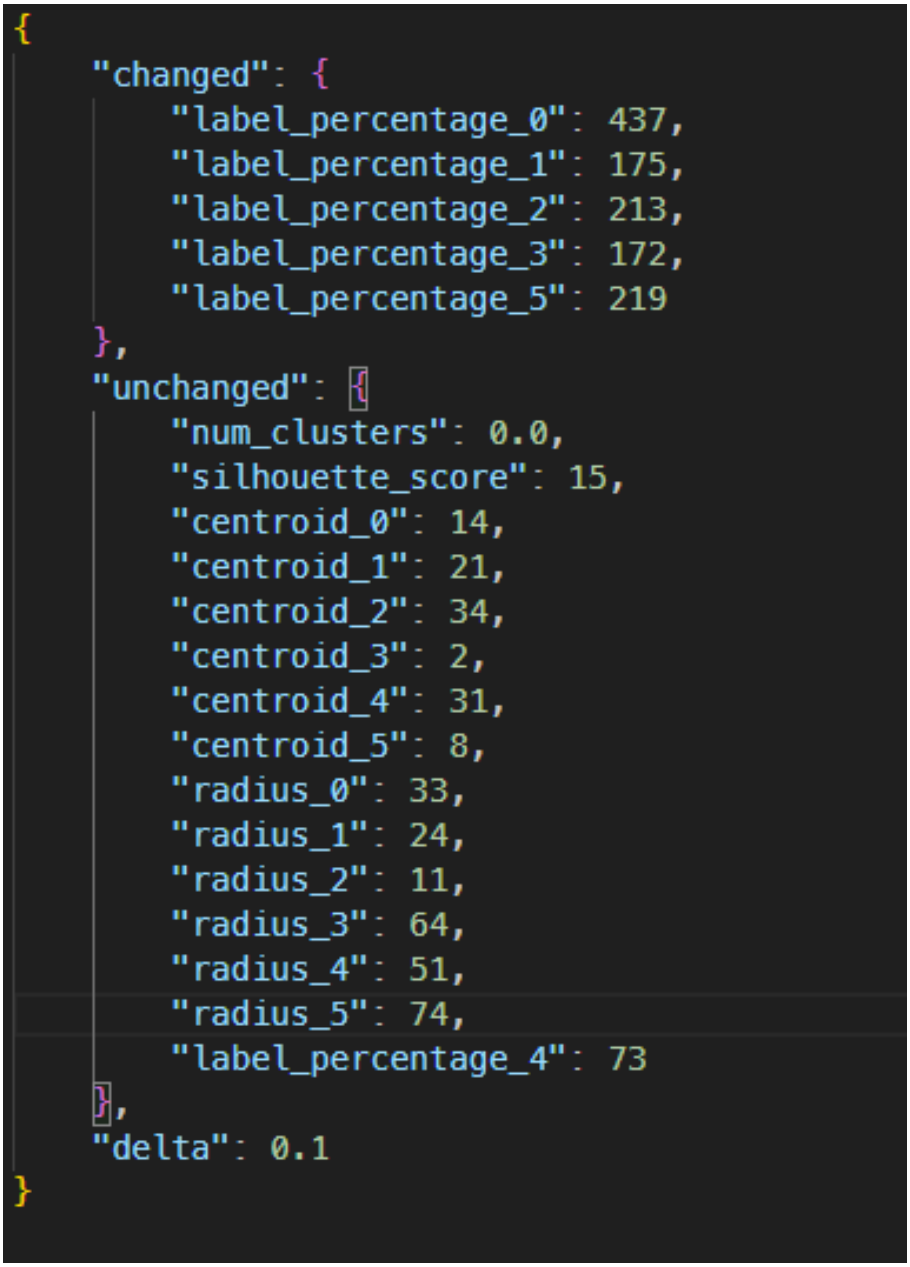

Figura 5.14: Data drift view in metrics

5.6. Retraining

I repeated the same process for good wine data and tested with poor quality wine. The metrics also dropped from around 70 % to a mere 50 %.

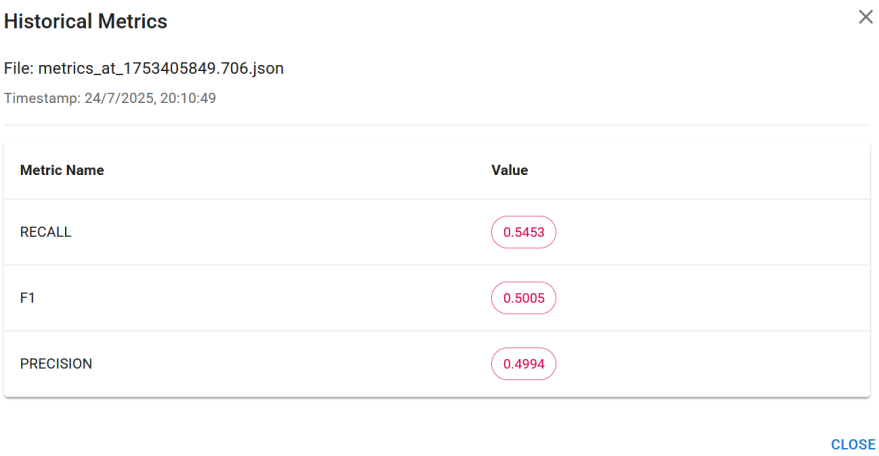

Figura 5.15: Data drift view in metrics reverse situation

In this situation, another solution is to use a combination of all data (full original dataset). The model selected has worst overall metrics, and is not such a specialized model. This means that the model will not be the best at differencing wines but will handle better the data drift as it knows good and bad wines. To test this, I will test it against the good and bad datasets and the SMOTEd dataset (artificial data) not seen by the model. In this situation, another solution is to use a combination of all data (full original dataset). The model selected has worst overall metrics, and is not such a specialized model. This means that the model will not be the best at differencing wines but will handle better the data drift as it knows good and bad wines. To test this, I will test it against the good and bad datasets and the SMOTEd dataset (artificial data) not seen by the model.

Capítulo 6

Analysis of impact

Throughout this work, a methodology has been developed to manage and automate ML projects in the best possible way. To do this, a review was carried out to understand why a methodology like MLOps is necessary and where it originates, what specific problem it aims to solve, the analysis of the tools that support the methodology, how the tool has been deployed, and the specific methodology to follow, illustrated through a practical project.

MLOps arises from the need to adapt agile development practices to ML developments. Although modern computing and artificial intelligence emerged simultaneously, the former achieved good results early on and was therefore more widely adopted. This led to a later emergence of development methodologies in ML. MLOps was created to address the same problems that DevOps tackles, but in the context of ML. MLOps has been shown to be more effective in managing the lifecycle of ML development than other techniques. Moreover, it is structured in levels, allowing the automation level to be tailored to each project. Full automation is not necessary for a project that requires only a one-off model, but it is essential for a project that demands consistent accuracy.

Among the available architectures, the most suitable has been selected for the presented problem. MLflow enables the management of ML projects at level 1. For deployment management, Docker has been used, as it handles dependencies with the operating system more easily and efficiently than a virtual machine, improving the system's replicability. PostgreSQL has been chosen to manage the supporting database, and the artifact repository is hosted on a cloud environment such as AWS or Google Cloud, accessed via S3 (AWS) and HDFS (Google Cloud). The addition of security and, specially, automatization (autodeployment, model monitorization, and metric evaluation) and notification to the users of all the automatization.

Once the tool and the architecture have been selected, the methodology to follow must be implemented. This methodology has been adapted to make the most of the tool, taking advantage of its strengths and minimizing its weaknesses. In this case, the flexibility of MLflow in terms of development is exploited: it natively supports multiple alternatives and provides basic support for the rest. Automatization could be implemented since training faces AutoML from platforms where it is openly available (like H2O).

Another pillar of MLOps is model replacement when their performance declines. To

detect this, it is necessary to store all queries made to the models. This methodology has been developed with the goal of providing traceability, reproducibility, and agility to ML projects. Traceability is ensured by recording everything related to the produced models, allowing experimentation with different parameters without fear of losing previous work, ultimately aiming to produce better models. Replicability has been ensured from the beginning, with multiple replicated server instances available to test concepts and architecture. The system itself supports replicability by saving all information related to how experiments and runs were performed, enabling them to be replicated at any time if necessary. The agility of the methodology is directly inherited from DevOps, as MLOps is an extension of it. Development and production environments are also replicated, in this case as training and production environments.

Regarding its relationship with sustainable development and the UN's goals, this project aligns with the ninth objective: "Industry, Innovation, and Infrastructure." MLOps is a methodology developed over the past eight years for project management, first introduced by Google [21], whose purpose is to streamline ML projects in order to achieve better results. Its use not only drives innovation but is also intended to serve research. It reduces the need to repeatedly perform the same training to obtain data, thereby lowering energy consumption in the most costly part of the process.

Capítulo 7

Conclusions

In this proposal, the objectives initially proposed have been fulfilled: MLflow was improved through the use of external tools, it was deployed on a Docker-based architecture, and a methodology was designed that follows MLOps:

- The MLOps application has been studied and used in a ML project.
- Improved automatization of mlflow from level 0 to 1.
- Design an architecture capable of automatization at level 1.
- Deploy and test the architecture.
- Notify the user of automatization triggers.
- Update the methodology to support automatization.
- Allow flexible deployment of models.
- Allow for agile and continuous delivery of models.
- Replicable creation of models.
- Retrieving data from models already in production.

Bibliografía

[1] Beck, Kent and Beedle, Mike and van Bennekum, Arie and Cockburn, Alistair and Cunningham, Ward and Fowler, Martin and Grenning, James and Highsmith, Jim and Hunt, Andy and Jeffries, Ron and Kern, Jon and Marick, Brian and Martin, Robert C. and Mellor, Steve and Schwaber, Ken and Sutherland, Jeff and Thomas, Dave. *Manifesto for Agile Software Development*. https://agilemanifesto.org/. 2001.

[2] Cray Research, Inc. *CRAY-1 Hardware Reference Manual (Publication No. 2240004 Rev. C, November 1977)*. Revision C manual for the CRAY-1 system, scanned from Bitsavers.org. Nov. de 1977. URL: https://bitsavers.trailing-edge.com/pdf/cray/CRAY-1/2240004C_CRAY-1_Hardware_Reference_Nov77.pdf (visitado 25-07-2025).

[3] NVIDIA Corporation. *NVIDIA A100 Tensor Core GPU Datasheet*. PDF document, NVIDIA website. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf.

[4] Chris Garry. *Apollo 11 Guidance Computer (AGC) source code for Command Module (Comanche055) and Lunar Module (Luminary099)*. Digitized AGC source code hosted on GitHub, from Virtual AGC and MIT Museum. 2020. URL: https://github.com/chrislgarry/Apollo-11.

[5] *ISO/IEC TS 13211-3:2025 — Programming languages - Prolog - Part 3: Definite clause grammar rules*. Technical Specification 83635, published June 2025 by ISO/IEC JTC 1/SC 22. International Organization for Standardization (ISO) e International Electrotechnical Commission (IEC). Jun. de 2025. URL: https://www.iso.org/standard/83635.html.

[6] Edward A. Feigenbaum. *Proposal to the Advanced Research Projects Agency for the Continuation of the Heuristic Programming Project*. Stanford Heuristic Programming Project proposal; accessed via Stanford Digital Repository. Stanford University. 1979. URL: https://purl.stanford.edu/vp226xm6367 (visitado 25-07-2025).

[7] Autonomous Vehicle International. *The Prometheus Project*. Feature article on the history and impact of the Prometheus Project in autonomous vehicle R&D. 2025. URL: https://www.autonomousvehicleinternational.com/features/the-prometheus-project.html.

[8] Reuters. *Trump announces up to $500 billion private-sector AI infrastructure investment*. By Steve Holland; originally published January 22, 2025. Ene. de 2025. URL: https://www.reuters.com/technology/artificial-intelligence/trump-announce-private-sector-ai-infrastructure-investment-cbs-reports-2025-01-21/.

[9] The White House Office of Science and Technology Policy. *Public Comment Invited on Artificial Intelligence Action Plan*. Request for Information (RFI) open through March 15, 2025. Feb. de 2025. URL: https://www.whitehouse.gov/briefings-statements/2025/02/public-comment-invited-on-artificial-intelligence-action-plan/.

[10] European Commission. *EU launches InvestAI initiative to mobilise €200 billion of investment in artificial intelligence*. Press release published February 11, 2025. European Commission. Feb. de 2025. URL: https://digital-strategy.ec.europa.eu/en/news/eu-launches-investai-initiative-mobilise-eu200-billion-investment-artificial-intelligence.

[11] OpenAI. *ChatGPT*. Official product overview page for ChatGPT, including features, plans, and supported models. 2025. URL: https://openai.com/index/chatgpt/.

[12] OpenAI. *Sora - Explore (ChatGPT text-to-video model)*. "Explore" gallery page showcasing AI-generated videos; access restricted to ChatGPT Plus/Pro users. Dic. de 2024. URL: https://sora.chatgpt.com/explore.

[13] Google DeepMind. *Gemini - Google's AI assistant and multimodal foundation model*. Official landing page for Gemini AI models and applications. Dic. de 2023. URL: https://gemini.google.com.

[14] Google DeepMind. *Veo - text-to-video model*. Official DeepMind landing page for the Veo video generation model. 2024. URL: https://deepmind.google/models/veo/.

[15] DeepSeek AI. *DeepSeek - Open-Source AI Models and Assistant*. Official homepage for DeepSeek AI models, including DeepSeek-R1 and DeepSeek-V3. 2025. URL: https://www.deepseek.com/.

[16] Anthropic. *Claude - AI Assistant for Work and Creativity*. Official homepage for Claude, Anthropic's AI assistant models including Claude 4 Opus and Sonnet 4. 2025. URL: https://claude.ai.

[17] Intel. *Multimodality: A New Frontier in Cognitive AI*. Intel Community blog post discussing the integration of multiple modalities in cognitive AI systems. 2022. URL: https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Multimodality-A-New-Frontier-in-Cognitive-AI/post/1407751.

[18] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[19] Rising Odegua. *How to put machine learning models into production*. Stack Overflow Blog post discussing practices and methods for deploying machine learning models into production environments. 2020. URL: https://stackoverflow.blog/2020/10/12/how-to-put-machine-learning-models-into-production/.

[20] VentureBeat Staff. *Why do 87% of data science projects never make it into production?* VentureBeat article discussing challenges in deploying data science projects into production. 2019. URL: https://venturebeat.com/ai/why-do-87-of-data-science-projects-never-make-it-into-production/.

[21] D. Sculley et al. «Hidden Technical Debt in Machine Learning Systems». En: *Advances in Neural Information Processing Systems (NeurIPS) 2015*. NeurIPS 2015 conference paper. 2015. URL: https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463e-Paper.pdf.

[22] Google Cloud. *MLOps: canalizaciones de automatización y entrega continua en el aprendizaje automático*. Artículo en el Centro de Arquitectura de Google Cloud

que describe técnicas para implementar e automatizar la integración continua (CI), entrega continua (CD) y entrenamiento continuo (CT) en sistemas de aprendizaje automático. 2024. URL: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=es-419.

[23] IBM. *Exploratory Data Analysis*. Overview article on EDA techniques, tools, benefits, and best practices from IBM Think. 2025. URL: https://www.ibm.com/think/topics/exploratory-data-analysis.

[24] Hugging Face, Inc. *Hugging Face — The AI community building the future*. https://huggingface 2025.

[25] GitHub, Inc. *GitHub — Build and ship software on a single, collaborative platform*. https://github.com/. 2025.

[26] McKinsey & Company. *Operationalizing Machine Learning in Processes*. McKinsey article outlining a four-step approach to embed ML into core business operations. 2025. URL: https://www.mckinsey.com/capabilities/operations/our-insights/operationalizing-machine-learning-in-processes.

[27] Amazon Web Services. *MLOps Automation With SageMaker Projects*. AWS documentation for SageMaker Projects explaining MLOps automation features. 2024. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-why.html.

[28] Matei Zaharia, Andrew Chen, Aaron Davidson et al. *MLflow: Open Source Platform for the Machine Learning Lifecycle*. https://mlflow.org/. 2018.

[29] Ben Wilson et al. *mlflow/mlflow: Open-source platform for the machine learning lifecycle*. 2025. URL: https://github.com/mlflow/mlflow.

[30] Docker, Inc. *Docker Engine Networking Overview*. https://docs.docker.com/engine/network/ 2025.

[31] Docker, Inc. *Use Volumes | Docker Docs*. https://docs.docker.com/engine/storage/volumes/. 2025.

[32] Docker, Inc. *Use Bind Mounts | Docker Docs*. https://docs.docker.com/engine/storage/bind-mounts/. 2025.

[33] Docker, Inc. *Compose file version 3 reference*. https://docs.docker.com/reference/compose-file/. 2025.

[34] Python Software Foundation. *venv — Creation of virtual environments*. https://docs.python.or 2025.

[35] Anaconda, Inc. *Anaconda.org: Package Hosting and Management Service*. https://anaconda.o 2025.

[36] DB-Engines. *DB-Engines Ranking of Relational DBMS*. https://db-engines.com/en/ranking/r 2025.

[37] MLflow Maintainers. *MLflow Releases*. https://github.com/mlflow/mlflow/releases. 2025.

[38] Stack Overflow community. *Data lost while restarting postgres with docker*. https://stackoverflow.com/questions/39808273/data-lost-while-restarting-postgres-with-docker. 2016.

[39] Alonso García Velasco. *MLOps-ResearchProject-TFM*. https://github.com/aloncrack7/MLOps ResearchProject-TFM. 2024. DOI: 10.5281/zenodo.17055908. URL: https://zenodo.org/records/17055908.

[40] Alonso García Velasco. *data-degradation-detector*. https://github.com/aloncrack7/data-degradation-detector. 2024. DOI: 10.5281/zenodo.17057825. URL: https://zenodo.org/records/17057825.