UNIVERSIDAD DE CASTILLA-LA MANCHA

BACHELOR IN COMPUTING ENGINEERING

Coursed intensification: Computer Engineering

BACHELOR DISSERTATION

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management

Miguel Ángel Ruiz Arreaza

Escuela Superior de Informática

Ciudad Real, Julio de 2025

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Tutor's Department:

Coursed intensification: Computer Engineering

BACHELOR DISSERTATION

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management

Author: Miguel Ángel Ruiz Arreaza

Tutor: Pablo Tomás Toledano González

Co-tutor: José Luis Espinosa Aranda

Julio, 2025

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management

This text has been prepared with the LaTeX template for Final Dissertations in Computing Engineering for UCLM, published by Jesús Salido in the public Zenodo repository, DOI: 10.5281/zenodo.4561708.

To my family, close friends, fellow colleagues, classmates and professors
For their patience and for accompanying me and showing me the way through this exciting journey.

TRABAJO FIN DE GRADO - ESCUELA SUPERIOR DE INFORMÁTICA (UCLM)

MADTrack: DS for Dataset and AI Model Management

Miguel Ángel Ruiz Arreaza
Ciudad Real, Julio 2025

Abstract

«What»

Artificial Intelligence is on the rise, and many companies are striving to create a wide-ranging variety of models to help themselves and their customers in an equally wide range of specific tasks. This inevitably translates into multiple model trainings being performed a year and, in many cases, many datasets being created or modified in the same time interval. Such is the case of UBOTICA Technologies: a Space:AI company that works in Computation *on the edge*, delivering AI solutions integrated in embedded systems incorporated in spatial modules, which come with limited space for storing data and computing power. The development of these solutions requires multiple training and deployment iterations, which over the years presented challenges in maintaining optimal traceability across their AI training environments, increasing the time required to search for a specific AI training configuration.

«How»

MADTrack is a distributed configuration management system with the purpose of putting order to the aforementioned challenges. It will store, track and manage all changes within datasets and AI model configurations. The main restrictions over the development of the system are the limited storage space of the company for this resources (the management of the evolution of datasets and models has to be done efficiently), the distributed nature of the environment where the items are stored and managed and the need for the system to be integrated in a greater processing workflow. The development of the system will be divided in a series of prototypes with an iterative and incremental approach, following continuous testing policies.

«Conclusion»

The resulting system will consist of a distributed system following the client-server application, with a local or remote server attending requests from multiple clients sending requests by means of a software wrapper library which in turn will interact with other open-source technologies.

Keywords: Escuela Superior de Informática, UCLM, TFG, LaTeX, TeXLive, AI, Deep Learning, Distributed System, Dataset, configuration management .

BACHELOR DISSERTATION - ESCUELA SUPERIOR DE INFORMÁTICA (UCLM)

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management

Miguel Ángel Ruiz Arreaza
Ciudad Real, Julio 2025

Resumen

«Que»

La Inteligencia Artificial está en ascenso, y muchas empresas están trabajando en la elaboración de una inmensa variedad de modelos que las ayuden tanto a ellas como a sus clientes a realizar una variedad de tareas igualmente amplia. Esto provoca que se realicen muchas entrenamientos de modelos al año y, en muchos casos, se creen o modifiquen muchos conjuntos de datos en el mismo intervalo de tiempo. Este es el caso de UBOTICA Tecnologías: una empresa de Space:AI que apuesta por la computación *on the edge* para ofrecer soluciones de Inteligencia Artificial integradas en sistemas empotrados en módulos espaciales, que tienen un espacio de almacenaje y poder computacional limitado. El desarrollo de estas soluciones requiere de varias iteraciones de entrenamiento y despliegue, lo que ha llevado a un desorden caótico de conjuntos de datos y modelos difícilmente identificables, lo que dificulta la búsqueda de un modelo o configuración de entrenamiento específica.

«Como»

MADTrack es un sistema de gestión de configuración distribuido que tiene el propósito de ponerle orden al caos configuracional previamente mencionado. El sistema almacenará, rastreará y gestionará todas las modificaciones de conjuntos de datos y configuraciones de modelos IA.Las principales restricciones que tiene el desarrollo del sistema son el espacio de almacenamiento limitado de la empresa para estos recursos (la evolución de conjuntos de datos y modelos debe ser registrada de manera eficiente), la naturaleza distribuida del entorno donde se almacenan y gestionan los elementos y la necesidad de la integración del sistema en un gran flujo de procesamiento. El desarrollo del sistema se dividirá en una serie de prototipos con un enfoque iterativo y incremental, siguiendo políticas de pruebas continuas.

«Conclusiones»

El sistema resultante consistirá en un sistema distribuido que sigue la arquitectura cliente-servidor, con un servidor local o remoto atendiendo las solicitudes de varios clientes enviando solicitudes a través de una biblioteca Software del tipo Wrapper que a su vez se interactúe con otras tecnologías de software abiertas.

Palabras clave: Escuela Superior de Informática, UCLM, TFG, LaTeX, TeXLive, AI, Deep Learning, Distributed System, Dataset, configuration management .

Agradecimientos

Aunque es un apartado opcional, haremos bueno el refrán *«es de bien nacidos, ser agradecidos»* si empleamos este espacio como un medio para agradecer a todos los que, de un modo u otro, han hecho posible que el trabajo realizado *llegue a buen puerto*. Esta sección es ideal para agradecer a directores, profesores, mentores, familiares, compañeros, amigos, etc.

Estos agradecimientos pueden ser tan personales como desees e incluir anécdotas y chascarrillos, pero recuerda que *no deberían ocupar más de una página*.

Miguel Ángel Ruiz Arreaza
Ciudad Real, 2025

Notación y acrónimos

NOTACION

(Texto aclaratorio -*suprime*-). Ejemplo de lista con notación (o nomenclatura) empleada en la memoria del TFG. Debes editarla según las necesidades de tu trabajo intenta que sea informativa y evita que incorpore información obvia.[1]

A, B, C, D	:	Variables lógicas
f, g, h	:	Funciones lógicas
$\cdot$	:	Producto lógico (AND), a menudo se omitirá como en AB en lugar de $A \cdot B$
$+$	:	Suma aritmética o lógica (OR) dependiendo del contexto
$\oplus$	:	OR exclusivo (XOR)
$\overline{A}$ o A'	:	Operador NOT o negación

LISTA DE ACRÓNIMOS

(Texto aclaratorio -*suprime*-). Ejemplo de lista *ordenada alfabéticamente* con los acrónimos empleados en el texto. Se pueden omitir aquellos acrónimos que son reconocidos en el contexto académico (p. ej., PhD), aunque aquí se han incluido a efectos ilustrativos.

[1] Se incluye únicamente con propósito de ilustración, ya que el documento no emplea la notación aquí mostrada.

Contents

List of Figures

List of Tables

Índice de listados

Índice de algoritmos

CHAPTER 1
Introduction

With the passage of time, the development of Artificial Intelligence (AI) has become of increasing interest due to the powerful tools it can provide to any organisation [6]. Since the development of *The Bombe*, the machine that was able to decode the *Enigma* machine in 1939, passing through a whole set of ups and downs and even a silent winter before its comeback in 2015 thanks to Deep Learning, Artificial Intelligence is being applied in a number of fields, sometimes even reaching the point of having the potential of threatening human safety and raising awareness of the need of regulations for its use.

The development of these models has been made possible thanks to the contribution of a number of organisations, such as Google, Microsoft and OpenAI, which invested an objectively significant amount of time and money (reaching a total investment of 24.0 billion dollars in 2018 [6]) to develop systems as broad as the GPT-4 model, which is able to generate text from both imaged and textual inputs [2], and is capable of helping professionals in solving doubts in a large number of fields.

With time, the development of these models became increasingly complex and difficult, and this would be reflected in the number of iterations required for a model to reach the desired level of accuracy in its predictions. Moreover, the application of these techniques on areas where little data was available made this development even harder. Some companies even started to employ their own resources to gather new data to train their models, which led to a model having more iterations as the available data grew. Summarizing, a model (and even a dataset) can become complex structures that have many evolutionary stages within their lifecycle.

Many organisations, aware of the increasing complexity of the lifecycle of datasets and models, saw a market opportunity to develop systems that would manage these lifecycles in an organized and efficient way, producing the AI and dataset configuration management tools, such as MLFlow [32], DVC, which were open source and available to everyone, as well as premium tools such as Neptune.ai, which are Platform as a Service (PaaS) that provide even further monitoring capabilities.

The aim of this Final Degree Project is to tackle this issue in a specific particular case. In the coming subsections, the reasons that motivated the elaboration of this project, along with the challenges it faces and how they will be solved, will be thoroughly described. Finally, an overview of the structure of the document will be given, so as to give readers the necessary information to follow the development documentation of the project.

1.1. MOTIVATION

It is of common knowledge that Artificial Intelligence has gained significant importance and visibility since the apparition of AlexNet in 2011, which was the basis for Deep Learning along with the possibility to use GPUs for training neural networks. The development of tools such as the GPT models have headed a revolution in the way humans solve both simple and complex tasks. A revolution that

was made possible with the intervention of multiple companies and organisations, and a considerable amount of money and time invested in the development of these models [6].

UBOTICA Technologies is a pioneer company that develops AI Computer Vision (CV) solutions. This means that, as a company, they use Artificial Intelligence techniques to extract information from images. These solutions are then integrated in embedded systems with limited capabilities, and that are part of a bigger, more complex system. The domains where these solutions are used are mainly in the space industry with their new CogniSAT-6 project [22], and recently even in the culinary industry. The development of the various solutions of the company requires multiple iterations where the models are trained, validated and tested, either in existing datasets, or in new ones. Moreover, the CogniSAT-6 system has the capability for creating new datasets out of self-taken images, which may be periodically added to the existing datasets, or even used to replace the existing ones.

This continuous rise of the available models and datasets, paired with a lack of a real control protocol over the new and improved versions of an AI model or dataset, has often led to situations where an abnormal amount of time is spent searching for the desired dataset, and accessing the necessary training configuration that produced a specific result.

The aim of MADTrack is to develop a system that establishes a robust configuration management basis for these datasets and AI models. A system that can integrate both new and existing datasets and models in a distributed, remote environment, and that will make the best use of the available resources the company dedicates to this management.

1.2. PROBLEMS AND SOLUTIONS

The main problem to be tackled in this Final Degree Project is the lack of a robust configuration management protocol over the produced datasets and models. Another problem is the difficulty at not only determining the correct configuration, but also bringing it to the environment where it is going to be used, and the need to adapt the system to the end users' existing methods to ensure familiarity.

Furthermore, another problem resides within the protocols and deployment requirements of the system, which may need special authentication protocols from the environment where the system is going to be deployed, as well as from the environment where the models and datasets are stored (which in turn may also differ from the one where the system is running). All of this may result in the need to develop a distributed system, where many components interact with each other to satisfy a complex need. Moreover, the system must be easy to integrate in bigger workflows, so the study of the available pipelines the company uses and the possible integration points of the system within them may be taken as another problem.

Finally, the last problem is the limitations of the company to dedicate resources for the storage of all the data produced by the model and dataset lifecycles. The developed system must, hence, make an efficient use of the resources available, so as to maximize the amount of data that could be stored and managed. Also, the system must be able to adapt to the needs of a growing company, where many requests may be taken concurrently.

For the resolution of all these three problems, the system will be formed by a server that will satisfy requests and perform the necessary storage and fetching operations to bring both the datasets and the models to the users, whilst the client will be a library that enables the integration of the necessary items into the configuration management, sending the evolutionary changes of a dataset or model to the server that stores them.

1.3. DOCUMENT STRUCTURE

According to the steps necessary to fully describe the elaboration process of this Final Degree Project, the following structure has been decided:

1. **Introduction**. The domain and main problematics are described, as well as what solutions the project will establish to these problematics.
2. **Objective**. The main objective, as well as the specific objectives of the project, are enumerated and detailed.
3. **State-of-the-art research**. This chapter contains the results of an extensive research and study of all the concepts and technologies relevant to the development of the project.
4. **Methodology**. Description of the development framework and methodology to be used, its specific application to the development of the project and the equipment used during it.
5. **General Configuration Management in MADTrack**. Development chapter focused on specific features of the project, particularly those in relation to General Configuration Management.
6. **Dataset Configuration Management in MADTrack**. Development chapter focused on specific features of the project, particularly those in relation to Dataset Configuration Management.
7. **Model Configuration Management in MADTrack**. Development chapter focused on specific features of the project, particularly those in relation to Model Configuration Management.
8. **MADTrack Tracking Server deployment**. Development chapter that plans the deployment of a specific component of the distributed system, and how does it integrate itself with the rest of the features.
9. **Results and tests**. Documents how the system was able to be integrated in a real workflow, the steps followed and the results obtained.
10. **Conclusions**. A summary of the achieved results will be made, as well as the proposal of the future work to be carried out on this project.
11. **Bibliography**. References to the works that have been used in the elaboration of this document.
12. **Appendix**. Sections with the auxiliary contents to better understand the project.

CHAPTER 2
Objective

The main aim of this chapter is to define the main objective of the project, and also enumerate and detail the specific objectives that will be necessary to fulfill so as to achieve it. The definition of objectives serves the pupose of providing a better explanation of the work that will be carried out within the scope of the project, and to provide an indicator of its good progress.

2.1. MAIN OBJECTIVE

The main aim of the project is to produce a system capable of tracking the configuration of datasets and AI models, and is easy to integrate in the internal workflows and pipelines of the end user (UBOTICA Technologies). The system will also be characterized by its distributed nature, its scalability (it will make efficient use of the resources, so that an increase on the resources or number of users will make a minimal impact on the system's performance), security (the system must be prepared for possible attacks, specially from injection and buffer overflow attacks) and robustness (Upon the case of minor failures, the system must remain operational and provide adequate, meaningful logging).

The system will provide user-friendly mechanisms for accessing the dataset and model configuration database, so the users can make operations on this configuration directly from their codespaces.

2.2. SPECIFIC OBJECTIVES

The aforementioned main objective can be divided into a set of partial objectives, also referred to as subobjectives. This final project can be divided into – subobjectives that will mark the development progress of the project, which could be in turn considered as finished when all of the subobjectives have been completed, and the resulting system satisfies the specifications of the main objective.

- *Development and deployment planning of a server able to track the configuration of AI models and datasets.*

  A planning will be made regarding the deployment details and the necessary infrastructure to be able to host the Tracking server that will satisfy the requests from the other components of the system. These details involve the necessary hardware requirements (Memory, CPU cores, network configuration, available ports) and software requirements (dependencies and entrypoint scripts) that will be used to design and develop the tracking server, which will be deployed in the future inside of the company's intranet infrastructure. It is also necessary to specify how this server will interact with the rest of components of the system and when should it be deployed so as to ensure its proper functioning.

- *Development of a library module that manages the configuration management of datasets. Registering changes on their contents.*

Aside from the tracking server, multiple library components will be necessary to manage the configuration of datasets and models. Some of these components will be developed under a module that will handle issues regarding the configuration management of datasets. These components will focus on providing code mechanisms that enable the integration of new datasets into the system, as well as providing the necessary means to bring the datasets in a specific evolutionary stage to the users.

- *Development of a library module that manages the configuration management of AI models, and facilitates the search of models according to their performance.*

  Other components of the library will be gathered within a module focused on managing the configuration of Artificial Intelligence models. These components will interact with the tracking server in order to track the parameters and metrics produced by the experimental runs of the models performed by the company, and register the final models and any other file meaningful to these inside a database.

CHAPTER 3

State of the art

This chapter aims to put into context the different concepts and technologies relevant to the development of the project. The following subsections provide an overview of these technologies, as well as the related concepts they handle, so as to contextualize the upcoming work. In the coming sections, concepts about Artificial Intelligence models, datasets and configuration management will be presented. Then, a thorough overview of the available technologies on the market to solve the problems described within Chapter 2 will be given.

3.1. DATASETS

In a statistical context, and according to a recent article by IBM employees [17], datasets are collections of organized data. The organization structure comes in a wide-ranging variety of formats, being the most common ones Comma Separated Values (CSV) and JavaScript Object Notation (JSON). The data contained within them can be collected from a number of sources, e.g. customer interactions, data obtained from distributed IoT devices (In the case of the end users, the satellites equipped with camera devices for taking pictures), or even public social media.

These arrangements of data are of great value for statistical data analysis, as well as for Machine Learning (ML) and other AI techniques, which require large amounts of accessible and reliable data to achieve a satisfactory performance. The quality of the data contained within a dataset, according to the definition standard stablished by the DEEL Workgroup in 2020 [9], is achieved by in turn achieving high ratings in several indicators, such as representativeness, traceability, accuracy, reliability and consistency, among many other.

With reference to how these datasets are used in Artificial Intelligence, these datasets are usually partially or fully labelled into one out of three categories [9]:

1. **Training datasets:** these parts of a dataset (or multiple datasets) will be used during the execution of the model's training algorithm. During the algorithm, the model's parameters are automatically adjusted iteratively until the former's convergence.
2. **Validation datasets:** Just like the training ones, these are the parts of a dataset(s) used to determine the generalization capability of the trained model To a dataset different than the one used in training. As a result, the data scientists will receive information about how the training hyperparameters should be adjusted.
3. **Testing datasets:** the data arranged under this category serves the purpose of serving as the test bed where the model's performance will be evaluated. This performance is most of the times operational, and the output of the testing phase are the model's performance metrics.

3.2. ARTIFICIAL INTELLIGENCE MODELS

An Artificial Intelligence model can be considered as a mathematical model which is capable of adjusting its parameters automatically and intelligently in order to reach a goal. This adjustment may be done based on the outcome previous iterations, also called *experience*. This learning may be assessed by agents external to the model (supervised learning) or may just depend on the model itself (unsupervised learning). There are cases in which the model assesses itself (reinforcement learning) [4].

There are several concepts that are of interest to this project that revolve around this concept of AI model, as their configuration will highly depend on this:

- **Parameters:** These are the terms of the described mathematical model. They are characterized by their mutability and the lack of external control of their adjustment during the training process. They are considered as the *black-box* elements inside a model.
- **Hyperparameters:** These are terms that express qualities on how the model will work. They are characteristically fixed and adjusted by data scientists prior to the training process.
- **Metrics:** Once a model has been trained and validated, the next step is to carry out an evaluation process using the testing datasets. After this evaluation, metrics are obtained. They are the main performance indicator for a model, since they explain how the model behaves on average for most of the datasets.

3.3. CONFIGURATION MANAGEMENT

> *"Configuration management is the discipline consisting in the unique identification, controlled storage, change control, and status reporting of selected intermediate work products, product components, and products during the life of a system."*

This definition by the *"Configuration management principles and practice"* book by Anne M. J. Hass [15] clearly establishes the important features of this discipline. The point of configuration management is to provide a traceable and accountable control protocol for any kind of system. In this case, our goal is to provide a way to uniquely identify and control the storage location of models and datasets, in order for the end users to spend less time searching for a particular dataset or a model with very specific parameter, hyperparameter or metric values.

Some of the ways this can be achieved are the use of version control systems, which allow users to establish different versions for a specific item; status checkers, which track the different states the item can be found in, or locator systems, which are able to determine the location of the item (specially useful in distributed environments, where the locations of the items may be physically separated by a considerable distance).

3.4. AVAILABLE TECHNOLOGIES AND TOOLS FOR AI MODEL AND DATASET CONFIGURATION MANAGEMENT

Now that the main concepts relevant to the project have been introduced, it is time to navigate the different solutions that are already available in the market which are capable of providing the mechanisms for AI model and dataset configuration management.

3.4.1. Git LFS

Git Large File Storage (LFS) is an open-source Git command line extension that enables users to manage and version large files in Git repositories by the means of a Git LFS server.

3.4.1.1. How does Git LFS work?

Git LFS reduces the overhead produced at uploading big files as blobs to a repository by storing the real contents of the files in a separate Git LFS server, while storing a special pointer file in the repository.

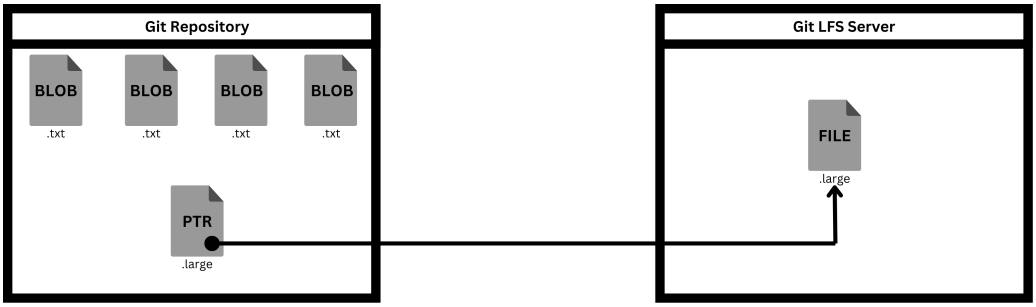

Figure 3.1: Explicative image that shows how LFS establishes a pointer to the remote LFS server.

Whenever a branch containing the large file is checked out, its contents are downloaded from the LFS server. In case the contents of the file are modified, the new version of the file is uploaded to the LFS server, which in turn generates a new version of the pointer file and makes a request to store it at the repository, all of it made in a transparent way for developers.

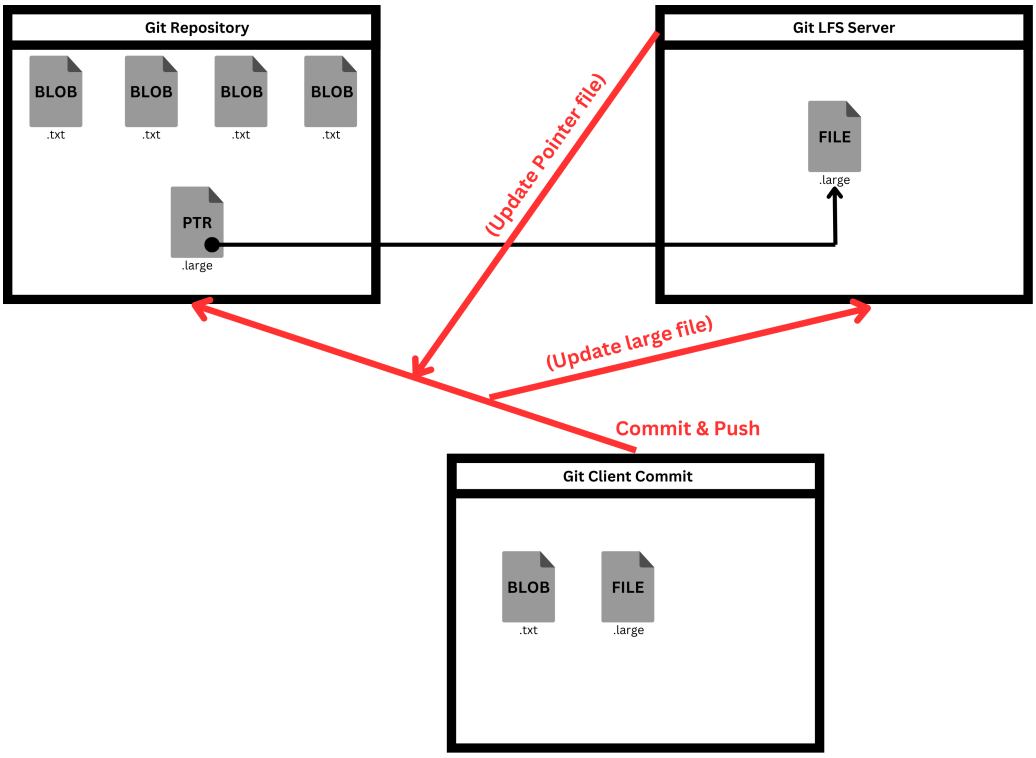

Figure 3.2: Diagram of the process LFS performs to carry out the commit operation over a registered large file.

With this simple mechanism, it is possible to transparently version and store large files in a Git repository without causing a huge overhead when performing operations at the Git repository.

3.4.1.2. Advantages and disadvantages of Git LFS

Advantages of Git LFS

- **High Performance**: The mechanism for storing large files (a few gigabytes large) in Git enables the management of these kind of files as well as not letting their size slow down the transactions or using too much memory.
- **Ease of use and familiarity:** The Git LFS command line interface is simple and easy to use for users already familiar with Git, given the similarity between the commands of both interfaces. This considerably flattens the learning curve of the tool.

Disadvantages of Git LFS

- **Not scalable:** If Git LFS were to be used in a distributed environment, it would not scalable due to the fact that it needs to be installed on every end user's system, and configured for every repository.
- **Limited in storage:** Git LFS has limited remote storage capacity (2 GB per file), falling thus too short on storage capacity for a project like MADTrack, which aims to handle datasets whose weight is counted in hundreds of gigabytes
- **Performance loss for very large files:** According to a Git LFS blog in Assembla [13], some performance losses were reported upon storing multiple very large files in a single Git LFS server. This serves as proof that Git LFS has difficulty in managing files heaver than a few gigabytes.
- **Difficult integration:** For integrating Git LFS into a system, it is also required to have Git Credential Manager installed (as a dependency). This dependence makes it a less modular solution than other possible alternatives.

3.4.2. Data Version Control (DVC)

Data Version Control (DVC) is an open-source, data-science-specialized software tool that provides mechanisms for easy data and experiment management, as well as for Machine Learning pipeline automation. It can be used in the form of an Interactive Development Environment (IDE) extension, a command line interface, or even as a library of the Python programming language. It is designed for aiding data science and machine learning teams.

3.4.2.1. How does DVC work?

According to its official documentation [25], the design of DVC is based on three main principles: codification (being able to define any important aspect of a Machine Learning project by making use of a metafiles, making room for best practices and engineering toolsets), versioning (it uses the Git workflow to enable teams to collaborate and share their work), and secure collaboration (enabling selective collaboration and access to all aspects of a project).

DVC has a very similar feel and workflow as Git, thanks to its operation on top of it. The storage space problem is solved by means of special files: DVC metafiles. This files have the dvc extension and serve as placeholders to track data files and directories. Additionally, there are other types of files, like the ones with the `dvc.yaml` extension, which have the purpose of modeling entire Machine Learning projects, or `.dvcignore` files, which are used to ignore certain files or directories. The files in DVC are stored in certain local or remote storage systems, but DVC treats them all under the name of DVC remote.

DVC handles repositories as DVC projects, which are initialized by the use of `dvc init`. Once a project is initialized, DVC offers a wide-ranging variety Of mechanisms to handle the configuration of the files and directories from a dataset. The `dvc add` command must be used for tracking an item.

Upon adding an item into DVC's tracking system, both a placeholder named `<filename>.dvc` will be created, as well as a `.dvcignore` file (if not existing yet) within the target's directory. The former will store the changes on the file or directory, whilst the other will prevent the original file or directory from being tracked by Git.

Furthermore, DVC is also able to retrieve or push data from or to a specific DVC remote using the `dvc pull/push` command, and to load a previous version of a tracked item after a checkout on the Git repository has been made, by making use of `dvc checkout`.

3.4.2.2. Advantages and disadvantages of DVC

Advantages:[11]

- **Specialisation:** DVC differs from other more general toolsets by its specialisation in the niche opened by emerging ML frameworks. DVC is aims to be used by Machine Learning teams for Artificial Intelligence and Data Science projects. Hence, the options provided by the tool easily adjust to the needs of these specialists.
- **Easy to learn:** DVC working on top of Git repositories and taking inspiration of their workflow makes it very easy to learn and understand. Most of DVC's commands are alike to those of Git.
- **High performance:** DVC has been designed to easily manage files the size of an actual neural networks dataset or a machine learning model. Hence, it makes it very efficient for these specific cases.
- **Compatibility:** DVC's remotes can be either local directories or remote storage systems. This may facilitate integration with the existing storage providers of the company.

Disadvantages:[11]

- **Slightly inflexible:** DVC will not accept an organisationally undefined environment in terms of architecture or design of the Machine Learning workflow. Teams must have a well-defined pipeline and metrics of the model in order to take full advantage of DVC's modeling tool.
- **Workflow imposition:** DVC requires a Git workflow. This makes it very difficult to use the tool in projects where this workflow is not well-suited.

3.4.3. MLFlow

MLFlow is an open-source project, specifically built with the objective of providing assistance in the Machine and Deep Learning process, managing the full lifecycle of projects of this nature and ensuring its traceability and reproducibility[27].

MLFlow offers mechanisms for Machine Learning model tracking and registry (handles configuration management), evaluation (it is able to track and compare different versions of a model) and reciping (provides guiding for structuring a Machine Learning or Deep Learning project).

This tool has gained particular popularity, since it is being used by the great companies, such as Microsoft, META and TOYOTA.

3.4.3.1. How does MLFlow work?

MLflow has a complex workflow consisting of various components, which are thoroughly explained in the following paragraphs [26].

The MLFLow tracking Application Programming Interface (API) is present in many programming languages (Python, R, Java and REST) and is part of the many components of MLflow, having the aim of providing the sufficient logging mechanisms for tracking a ML or DL project, including logging of parameters, code versions, metrics and output files.

In figure 3.3, an overview of the most common MLFlow deployment architectures show the different possibilities MLFlow can be used:

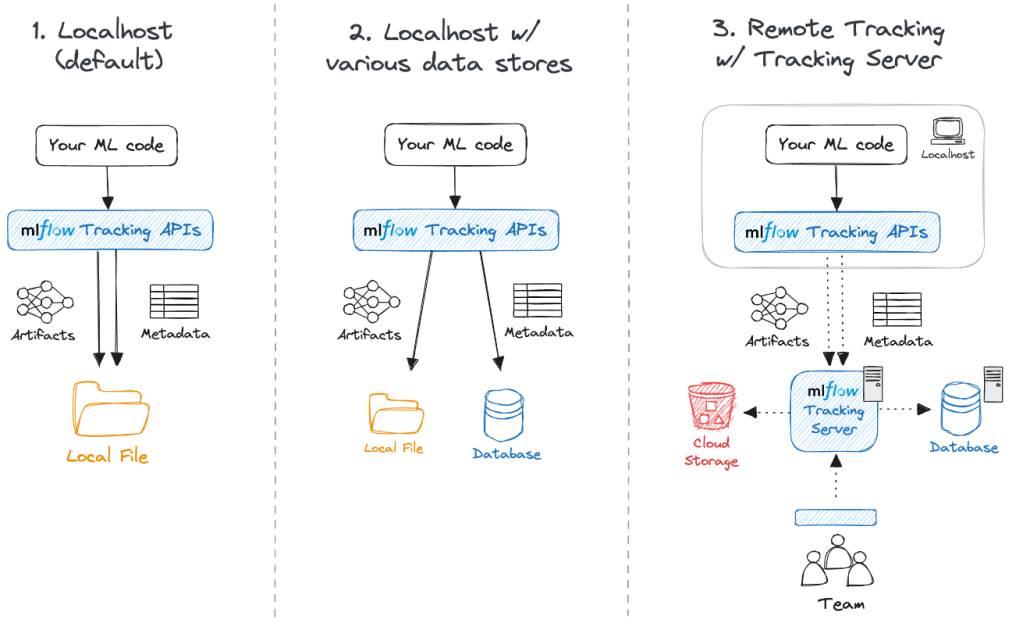

Figure 3.3: Diagram showing the different use scenarios of MLFlow. Source:[26]

In total, three use scenarios can be identified:

- **Localhost scenario:** MLflow stores and logs ML model metadata and artifacts inside a local file system directory.
- **Localhost with data stores:** Similar to the previous setup, but it also stores and logs data files in a remote file system or database.
- **Use of a remote tracking server:** The ML code is tracked by the API into a remote HTTP tracking server, which stores it in a remote storage and allows storing and retrieving items without directly accessing the storage media.

Apart from the API, there are more relevant components, such as the model registry. MLFlow's model registry provides the MLflow toolset with mechanisms for model lineage, versioning, aliasing, tagging and annotation. This component of MLflow can be accessed by running a proper MLflow server and using a backend storage medium. Whenever a model is desired to be registered, it is needed to be logged first. Some basic concepts to understand how this registry component works are:

- **Registrable model:** a model created from an experiment or run and logged with one of the model methods. Currently, MLFLow supports SKLearn, Pytorch and TensorFlow models.

- **Registered model:** a model registered with using the model registry. It contains a unique name, versions, aliases, tags and metadata.
- **Model version:** representation of a stage in the evolution of a model. Any newly registered model is assigned version 1.
- **Tag:** quality or set of qualities expressed about a model, expressed as a key-value pair. Allows model categorization.
- **Alias:** is a mutable, named reference assigned to a model version.
- **Annotation:** Markdown styled comment on a model in any model version. It used for describing and expressing relevant information to the project's collaborators.

MLFlow also provides its users with a friendly User Interface (UI) that is available on the cloud, and accessible by means of an internet browser. Before coming in detail about the provided functionalities, some concepts may be introduced first:

- **Model run:** process that englobes the training, validation and testing of a model.
- **Experiment:** representation of a set of model runs that share a common purpose, e.g. An experiment for ship segmentation models.
- **Artifact:** a file whose content is relevant to a certain model run.

This interface allows the navigation through the different registered experiments using a stack panel, as well as a comparative stack view of the different model runs, along with the parameters, metrics and datasets obtained within each run, which can be sorted according to the users' needs. The interface also provides menus for visualizing the different models and artifacts within a model run.

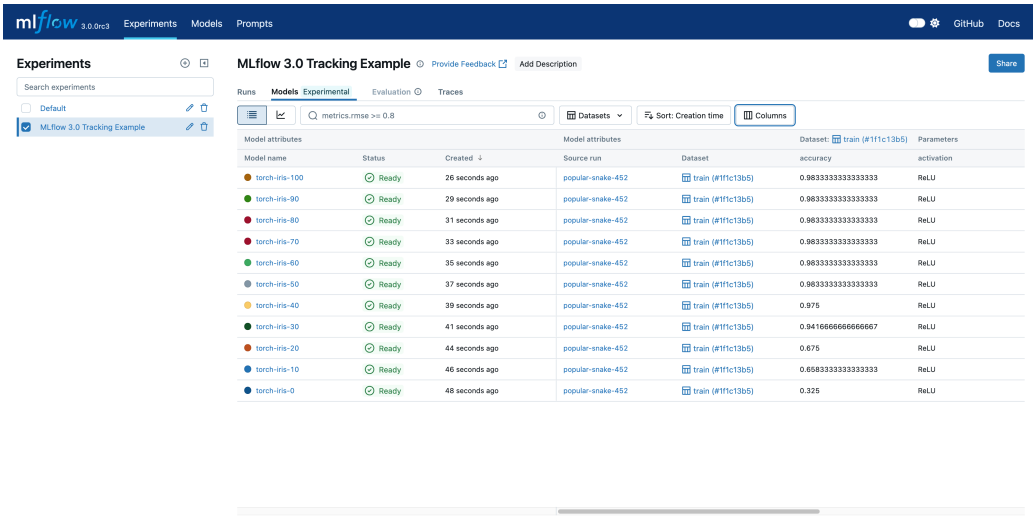

Figure 3.4: MLFLow's UI showing the list of runs contained within an experiment. Source:[26]

3.4.3.2. Advantages and Disadvantages of MLFlow[5]

Advantages:

- **Complete:** this tool excels at managing ML and DL models, with all the complexity it takes for tracking their evolution, versioning and even comparing them with other models.
- **Easy to learn:** apart from all the complexity hidden for managing the components of the tool, MLflow has an easy installation method and an intuitive UI. The documentation provides

users with a wide-ranging set of quick tutorials that will help them get started in the minimum amount of time.

- **Flexibility:** in the documentation, it is possible to see many tutorials which import libraries libraries related to the Machine Learning toolset with MLflow libraries, showing its versatility and integrability with many other toolsets related to Artificial Intelligence development. A quite desireable quality for a tool with this role inside a ML project.

Disadvantages:

- **Insufficiency in dataset management:** MLflow may be a great tool for managing the whole lifecycle of projects and AI models, but it lacks of mechanisms this good for dataset configuration management. This means that either way, another tool that manages specifically the datasets' lifecycle is needed, leading to integration, which leads in turn to higher complexity.
- **Difficult to dominate:** MLflow provides its users with many tutorials to give them a sense of how this tool works in record time. Nevertheless, the fact that the tool has a steep learning curve is undeniable. Its varied set of components makes it a complex tool with many possibilities, and it is difficult to gather the knowledge to master them all.

3.4.4. Neptune.ai

Neptune.ai is a PaaS that allows users to easily supervise and monitor AI/ML foundation model trainings and lifecycle. This set of functionalities classify under the definition of an **experiment tracker** application category. The features offered by Neptune include a combination between a database and and a dashboard, with logging support for machine learning models for Python, mostly like MLflow does, but adding the incorporation of a decent system for dataset versioning.

3.4.4.1. How does Neptune.ai work?[28]

Neptune.ai's system consists of two main components:

1. **Neptune.ai's Python library:** an API that can be found inside a Python library, used for logging and querying metadata extracted from model building. The tool also offers a Create-Read-Update-Delete (CRUD) interface for users and workspaces, as well as various monitoring mechanisms.
2. **Neptune.ai's web application:** a web application that enables users to visualize, compare, monitor and collaborate in AI projects.

The interactions between different clients and servers can be described within figure 3.5. As shown, various clients that have Neptune's library installed make request to an active Neptune Server, where the web application is also deployed in. The interaction between the users and the web application is regulated by the use of the Python library, creating a workspace for every existing organization, and a project per Machine Learning task.

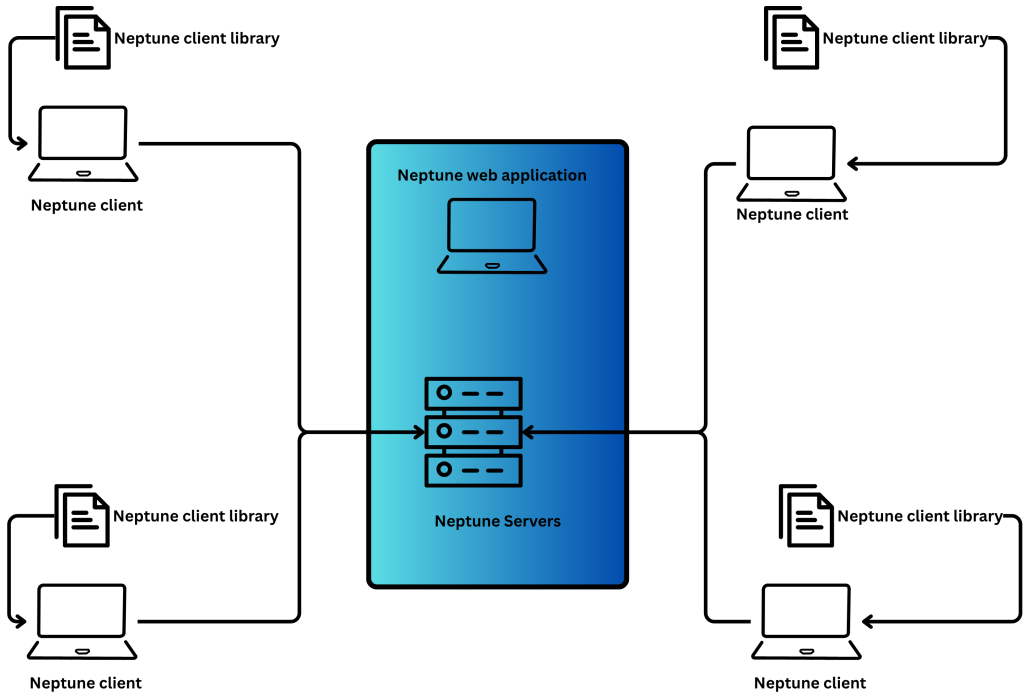

Figure 3.5: Interaction between Neptune clients and a Neptune server.

In order to better understand Neptune's environment, some concepts shall be introduced:

- **workspace:** a space for projects and team members with a specific storage amount, created with or without permission for public projects, and with one or more (non-human) service accounts. The storage limits of a workspace can be changed by the workspace administrators, but it may come with a fee charge (The free plan of Neptune only provides 200GB storage).
- **Service accounts:** privileged Neptune accounts that do not belong to human entities, and have the purpose of automating tasks that are normally performed through a human account. These accounts are workspace-specific and controlled by administrators.
- **Project:** represents the concept of a Machine Learning task. It registers all the tracked runs and shows them to all authorized members, allowing exploration and analysis. Projects have three visibility levels: Workspace (grants access for any of the project's workspace members), Public (available to anyone in the Internet) or Private (available only to the chosen members).
- **Run:** a tracked experiment of a project. It contains model-building metadata. They are created every time a model is trained or retrained and each time a model performs an inference by means of a executable script.
- **Model:** represents an AI Model. Is a collection of metadata common to all the model versions of the same model.
- **Model Version:** represents the version of a Machine Learning model. It consists of metadata common to a single version of a model.

Neptune also integrates with most of the Machine Learning toolsets available in the market. Making it feasible for integration in the project concerning this Final Degree Project. It counts with logging callbacks in order to easily integrate it around any model fitting.

Finally, Neptune.ai only takes a little knowledge of environment variables and terminal commands in order to run its UI, from which all the logging and metrics can be consulted and the metadata and performance can be easily visualized.

Once the main components of Neptune have been introduced, a generic workflow would have the following steps:

1. Add some lines of code before training a model. Just the ones for importing the right libraries and setting the logging callbacks.
2. Use the Neptune.ai web application to browsed the logged metadata during the training.
3. Visualize and compare run metadata.

3.4.4.2. Advantages and disadvantages of Neptune.ai[31]

Advantages:

- **Scoped for project needs:** Neptune.ai manages both models and dataset versioning, so no additional integrations are needed.
- **Designed for collaboration:** Neptune.ai is designed to be used in a collaborative environment, so it easily tracks the progress of a project made by any team member.
- **Standardized:** this tool complies with SOC2[14], a standard for security and compliance.
- **Ease of use:** Neptune.ai is a Python API package that can be easily installed and used in any Python project. It also allows the easy visualization the metadata of models and their training performance by using the web application.

Disadvantages:

- **Non-scalable:** the tool is not scalable to large datasets as the storage is limited, at least, in the free plan.
- **Limited by language:** The tool only provides a single Python package, which is incompatible with other languages.

3.4.5. Comet.ml

Comet.ml is another PaaS oriented to AI model evaluation, experiment tracking, and production monitoring, with particular expertise in Large Language Models (LLMs). It is used by companies like UBER and NETFLIX. It also provides a meeting point for all the ML development platforms (and even those not aimed exactly at developing this kind of models) and cloud infrastructure supported by providers like Amazon Web Services, Google Cloud and Azure.

3.4.5.1. How does Comet.ml work?[24]

Comet.ml is available as a Python package, a command line extension, and also provides an UI for visualizing and managing information on experiments.

With the python package,Comet enables logging for three types of information: AI model metadata (key-value pairs, metrics, parameters, information about the system, and so on), Assets and data (unversioned files like images, models, confusion matrices), and artifacts (versioned assets, like datasets). Comet.ml will track all relevant loggable assets until the code execution is over.

The user interface provides an option for customizing dashboard views, as well as for viewing and comparing experiments. It well combines with popular libraries like Scikit-Learn, Optuna and the self comet optimizer, helping users to find the optimal set of hyperparameters for their models. Comet also offers a model registry to track and version models.

Comet also introduces some concepts such as organization, workspace, project and Model Product Monitoring (MPM). Most of these being equivalent to the definitions provided in earlier subsections.

3.4.5.2. Advantages and Disadvantages of Comet.ml[30]

Advantages:

- **Ease of integration:** Comet is easy to integrate with other services and platforms, like Python Flask.
- **User-friendly interface:** Comet has been recommended by some G2 members as a tool with a very intuitive user interface.
- **Suited for collaboration:** Comet is designed for collaborative environments, where multiple members work at a time.

Disadvantages:

- **Costly:** The free plan of Comet.ml is very limited, while the premium plans scale up from 50 dollars a month.
- **Limited Customization:** Comet.ml does not provide as much customization options as other tools, which may be disappointing for the price of the service.
- **Limited by language:** The exclusive availability of the tool in the Python programming language restricts its flexibility.

3.4.6. Weights and Biases (WandB)

Weights and Biases, also known as *WandB*, is a PaaS dedicated to provide data scientists and AI developers with a toolkit that facilitates the production process of Artificial Intelligence models. It provides support for experiment tracking, hyperparameter sweeping, model registry and workflow automation, among many others.

3.4.6.1. How does WandB work?

WandB makes use of a set of lightweight, interoperable tools for managing the configuration of AI models, as well as for storing, displaying and sharing results. This makes the platform highly suitable for teams. It is available as either as Python and Javascript libraries, a Command Line Interface (CLI) and an interface that functions with an interface specialized in data addition.

Like most platforms, The main workflow consists of adding some code from the libraries prior to carrying out the training process, which connect with the cloud platform, for later tracking and visualization.

Some of the functionalities offered by WandB are:

- **Experiment tracking:** it allows to track the progress of a project made by any team member.
- **Sweeping:** the platform itself lets developers visualize the impact of the hyperparameters on the performance of the model. The platform is also able to perform open-source algorithms to infer the best possible values for the hyperparameters, visualizing all past and suggested configurations of the hyperparameters over a dashboard that the team can visualize.
- **Model Registry:** provides a way to govern over the trained models over their whole lifecycle.

- **Workflow trigger automation:** it offers tools for streamlining the learning pipeline and implementing sophisticated Continuous Integration/Continuous Deployment (CI/CI) processes by means of automatic workflow execution.

3.4.6.2. Advantages and disadvantages of WandB

Advantages:

- **Familiarity:** while gathering information to transform it into requirements, process detailed in coming chapters, it was revealed that some of the end users of this project already using this tool.
- **Versatility:** WandB can handle most (if not every) of the steps of the Artificial Intelligence models' lifecycle.
- **User-friendly:** WandB was reported in an article from Netguru[20] as a tool with a very intuitive user interface.
- **Code efficient:** The platform itself requires little amounts of code to be able to carry out the complete WandB workflow.
- **Suited for embedded system integration:** WandB can generate reports on power consumption and efficiency over the training and inference processes of an AI model, which can be very useful for embedded AI developers like the intended end users of the project.

Disadvantages:

- **Limited by language:** WandB is only available in Python and Javascript, and not in other languages.
- **Costly:** the free plan of WandB is very limited, while the premium plans scale up from 50$/month.

3.5. ADOPTION OF AI AND DATASET CONFIGURATION MANAGEMENT TECHNOLOGIES

This page provides the resulting overview from the exploration of other areas or companies that have already adopted a machine learning and dataset versioning platform, if possible in a similar context the one intended for this Final Degree Project.

3.5.1. Microsoft

In *section 3.4.3*, it was explained how MLFlow was used by various widely known companies, one of these being Microsoft. This company has indeed a section explaining how this tool works in their system and how they use it for integration with another tool: Ray[19]. Both of them are integrated as libraries in Python, and Ray tasks can be logged using the functions offered by MLfLow, thus proving the ease of integration that characterizes MLFlow.

3.5.2. WIX

The cloud website infrastructure and PaaS provider WIX is a company that, similarly to Microsoft, has chosen to use MLFlow as part of the tools integrated in the platform (as stated in *section 3.4.3*). They were specially interested in the integration of MLflow and Apache Spark to produce their very own Machine Learning platform, taking advantage of the model registry component. A deep study of how WIX uses this platform is described in a Medium story[12].

3.5.3. Edge Impulse

In 2019, the company Edge Impulse released the platform with the same name into the market. This platform provides a near-to-full range of tools to support Machine Learning Operations and Deep Learning pipelines adapted to TinyML, which are embedded systems specialized in or designed for running Machine Learning models. The platform provides a set of visual interfaces to make Machine Learning accessible to anyone. Some of its features, like the EON Tuner, also offer support for automated operations within the pipeline, thus assuring a more efficient and effective workflow. Up to now, the only limitation or important feature this platform lacks is the IoT device manager and production monitor[16].

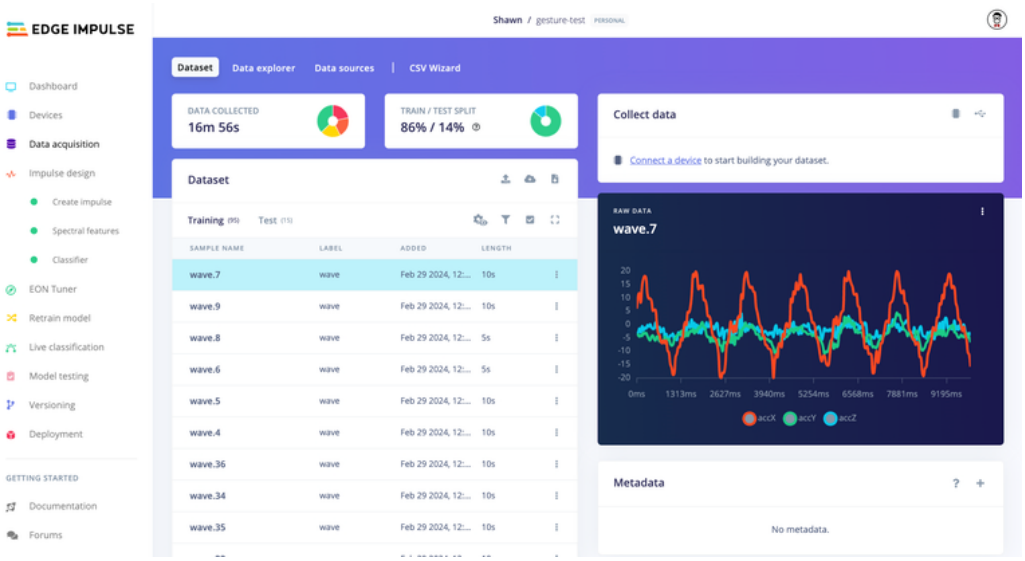

Figure 3.6: A look at Edge Impulse's UI. Source:[18].

CHAPTER 4
Methodology

This chapter contains details about the followed methodology during the development stages of this project, a time planning for the development stages, the resulting application of the methodology to this project, and the equipment (software and hardware) used during the development.

4.1. SELECTED METHODOLOGY

The selected methodology for the development of this project has been based on the Dynamic System Development Method (DSDM), performing some minor adjustments so that the methodology ties in with the project's needs.

This is an agile framework for software development. It was first published in 1995, with the aim of describing a methodology focused on quality, where the techniques for Rapid Application Development (RAD) could be also applied in. The main feature of DSDM is the use of prototyping techniques to guarantee the frequent delivery of software products to the end users[10].

In terms of plannification, most methodologies select to establish a former definition of the desired functionalities to be developed, to then establish a latter time estimation for the development of each functionality. DSDM, on the other side, focuses more on establishing a time estimation for the development of the project, in which then the amount of introduced functionality within the estimated time will be then organized[1].

4.1.1. Principles of the selected methodology

The DSDM has eight basic principles on which it is sustained. These are:

1. Focus on the business needs.
2. Deliver the artifacts and products on time.
3. Collaboration among team members.
4. Build upon a strong basis.
5. Iterative development.
6. Continuous and clear communication among stakeholders.
7. Control during the development of the project.

4.1.2. Techniques of the selected methodology

Throughout the course of the development of the project, DSDM makes use of a wide-ranging set of techniques, that may be used either alone or in conjunction with others according to the situation. The most important techniques (which are in fact used in some way in this project) are shown within *figure 4.1*, and detailed within the following subsections.

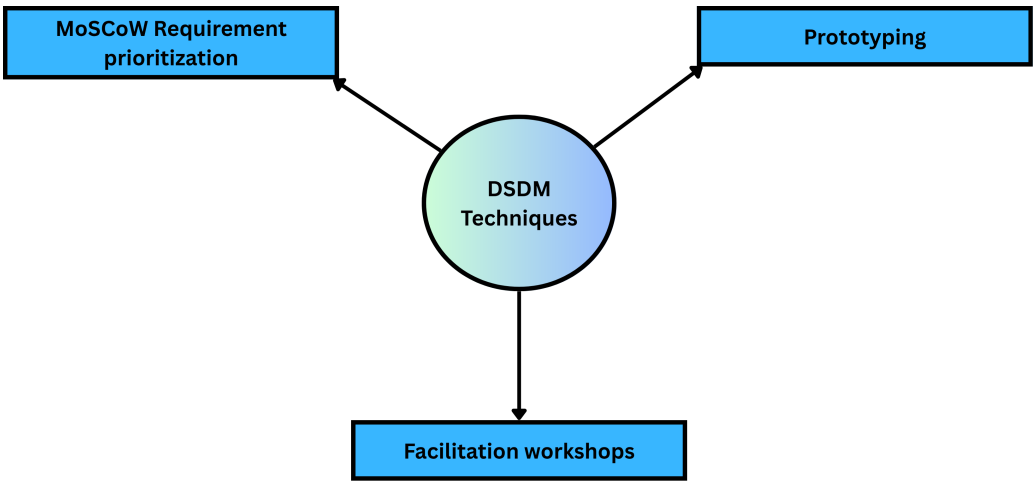

Figure 4.1: Spider diagram containing techniques widely used in DSDM.

4.1.2.1. MoSCoW requirement prioritization

The MoSCoW prioritization technique was created by software development consultant Dai Clegg in 1994, when teams at Oracle were using RAD and needed a quick method to establish requirement priorities. Some years later, in the 2000s, it became a popular technique specially in DSDM[3].

Prioritization is an essential part of the planning phase of every project. It establishes which of the collected requirements will provide end users with the most value, and should hence be worked on first. This prioritization is done classifying every single requirement under one of these categories: *Must have*, *Should have*, *Could have* and *Won't have* (*figure 4.2*). This classification gives all team members strategic knowledge about the impact of completing (or not) a requirement.

Figure 4.2: MoSCoW prioritization establishes four different priority levels for a requirement.

- **Must have:** they are paramount to achieve the project's success. Requirements classified under this category will be the first ones to be completed, since they have the greatest importance.
- **Should have:** while they are not essential, they also hold great value for the project, so they should be developed whenever possible.
- **Could have:** these requirements are mostly an additive to the baseline of the project, which would increase the satisfaction level of the end users. They will only be developed once the baseline is completed, and there are enough time and resources to dedicate to them.
- **Won't have:** this category contains requirements that may hold any importance within the project, but because of limitations on time and resources, they will be discarded from the scope of this project's phase. These requirements could be set to be developed in the future phases.

4.1.2.2. Prototyping

In order to have a shorter delivery interval, DSDM uses evolutionary prototyping techniques for the development of user requirements.

The main aim of evolutionary prototyping is to carry out a software development production process in an iterative and incremental approach. This means that multiple iterations of the software development lifecycle will be performed, each of them producing an increment of the system. Although the distinction between products and prototypes in this technique is fuzzy, the result of the first iterations will be considered prototypes, since they are not near to the full functionality of the final target system[7].

4.1.2.3. Facilitation workshops

These workshops are essentially meetings in which the stakeholders of the project meet with a specific objective related to the completion of deliverables. The purpose of the workshop is to reach clear solutions to the stated problems.

The development of these workshops is in many ways benneficial for the team:

- The workshop is an ideal environment for brainstorming, as well as for discussion and develop-men of the produced ideas.
- All the stakeholders get knowledge on the decisions made towards the project.
- Since all the stakeholders participate in the workshops, they will be able to propose ideas and take part in the decisive process.
- The decisions taken during the workshop are quick and precise.

4.1.3. Project lifecycle in DSDM

The DSDM methodology divides a project into seven phases to be completed sequentially. These phases are shown within *figure 4.3*.

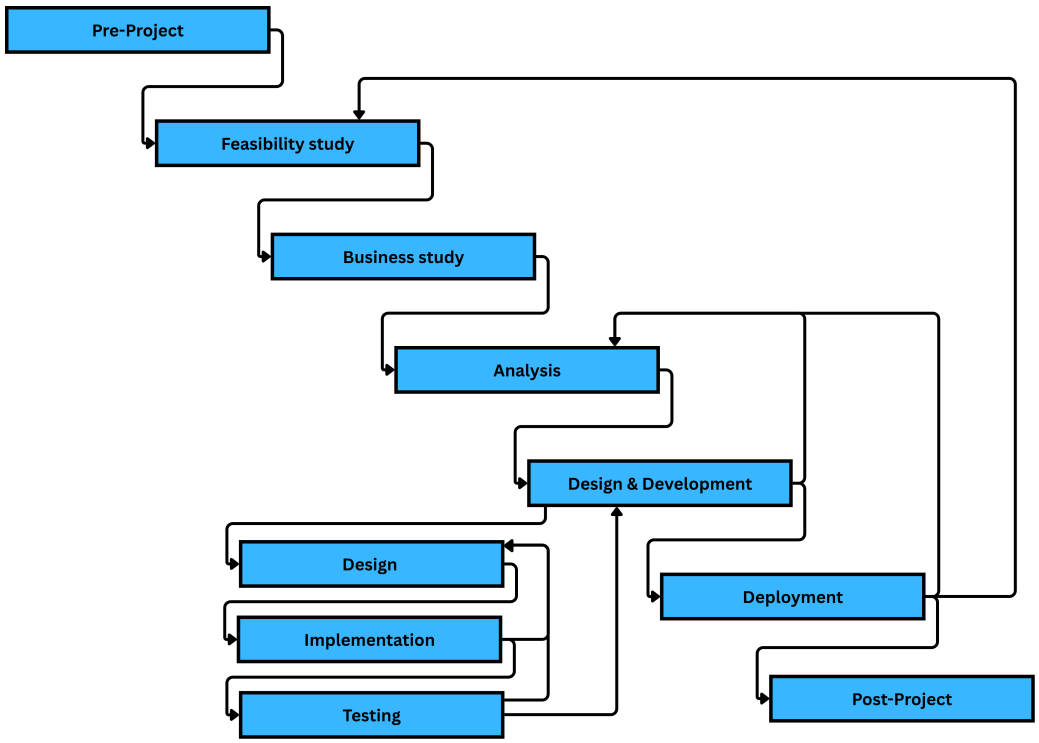

Figure 4.3: The lifecycle of a project, according to DSDM-based methodologies.

- **Pre-project:** this phase has the aim of verifying the establishment of the project over a real need. As well as diagnosing the availability of resources to perform a feasibility study.

- **Feasibility study:** apart from evaluating the adequacy of the use of DSDM for the development of the project, the techniques to be employed are decided and the balance between opportunities and risks upon development is presented. The outputs of this phase include a feasibility report and a provisional project development plan.
- **Business study:** the main aspects of the domain is analyzed, as well as the pontential technologies to be used. The affected business processes are identified, along with the components of the target system and the architecture of the system. After this, an initial prototype (which can be far different from the final look of the target system) is then created.
- **Analysis:** The first phase of the iterative and incremental development loop. The outputs include a prioritized requirement list, the analysis report made on those same requirements, and a risk analysis.
- **Design and development:** the iterative phase where the system is properly built. The low-level component interfaces are designed, to then provide an implementation of the system. This phase iterates until obtaining a target system that complies with, at least, the minimal specification formed by the requirements previously agreed.
- **Deployment:** This is the phase englobing the process by which the developed system is integrated within the end user's infrastructure, and consequentially delivered to them. On integration, the end user(s) receive a user's manual with all the information needed to correctly make use of the system. Depending on how complex this integration is, it is possible to carry out this phase iteratively.
- **Post-project:** After the project ends, an evaluation is made based on whether the specified objectives were satisfactorily fulfilled, and a list of lessons learned is elaborated. If some requirements were not fulfilled within the specified time, they could be left for future work on another project.

4.1.4. Roles in DSDM

The DSDM methodology defines fifteen different roles for all the stakeholders of a project. Below, there is an explanation of the most important ones:

- **Developer:** the role assigned to every member of the development team. That is, analysts, designers, programers and testers. These people directly participate in the creation of the system, specially within the *Analysis* and *Design and development* phases.
- **Technical coordinator:** assigned to those with the responsibility of guaranteeing the quality of the project and defining the architecture of the target system.
- **Visionary:** assigned to that team member that better understands the business objectives. They have the responsibility of ensuring the rapid completion of essential requirements (which according to the MosCow prioritization correspond to the ones labelled as *Must Have*), as well as supervising that the project goes well.
- **Executive sponsor:** role assigned to the stakeholders belonging to the end users' organization, and holds the financial power and authority over the project. Stakeholders with this role hold great importance in decision-making.

4.2. APPLICATION OF THE SELECTED METHODOLOGY TO THE PROJECT

This section takes the techniques, phases and roles enumerated and detailed in the previous section to then provide a summary of how and to what extent were they applied to the project. On first place, the application of the aforementioned roles will be detailed, to then make a summary of the obtained

requirements and their acquirement method, their classification following the MosCoW technique, and how iterations will be classified according to the represented progress type. Finally, the tentative time estimation per iteration will be presented, along with the actual duration of the iterations.

4.2.1. Roles assignation

According to the role definition made within *section 4.1.4*, the roles within the project have been distributed according to the principles defined in DSDM. The resulting assignation is the following:

- **Developer:** the only person to which this role has been assigned is the author of this document, Miguel Ángel Ruiz Arreaza. He is responsible of compiling the project's requirements, as well as building the target system. He is considered as multidisciplinary, carrying out the responsibility of an analyst, a designer, a programmer and a tester.
- **Technical coordinator:** this role has been assigned to the tutor of this Final Degree Project. He will be in charge of coordinating the development and integration and to guarantee the technical quality within this project.
- **Visionary:** the co-tutor of this project has been assigned this role. He will be in charge of coordinating the development team with the executive sponsor, so as to assure the alignment and the development of this project with the business interests and objectives.
- **Executive sponsor:** the assignment of this role goes to the CTO of the end users' company, which has the responsibility of providing the necessary resources to develop the project.

4.2.2. Requirement collection

The following subsections detail the methodology employed to obtain information about the current state of the configuration management procedures of the end users, as well of their preferences, needs and working methodologies. Once this methodology is explained, the results obtained from applying it will be described.

4.2.2.1. Requirement collection method

This project has the purpose of creating a system designed to provide value to the target company, which is UBOTICA Technologies. Hence, it is important to consider the needs of the employees that work with both ML and DL models and datasets, and who have to deal with the difficulty of not keeping track of the state of these. For this purpose, a set of interviews were conducted from November 18th,2024, to November 25th,2024. The selected sample was a group consisting of six employees working in AI and CV tasks close to production environments. Additionally, four of the interviewees also affirmed to carry out research tasks in environments far from production.

4.2.2.2. Requirement collection results

After conducting the interviews and taking important notes, these are the most important points that should be taken into account for producing a requirement specification:

- The end users work at CV, so the predominating dataset type is the image dataset. The system must do well at managing, moving and changing these.
- There is not any official set of important metadata, that are always present within every model or project. The model and project's necessary metadata changes with the purpose of the project. This means that the system must be flexible on the amount of metadata to introduce for a certain AI model or dataset.

- Hyperparameters take a particularly important role in model trainings for the end users, since they define important measures, like the length of the image bands and the number of colour channels. The system must consider a special form of classifying hyperparameters upon the training of a model.
- Upon having two versions of an AI model, many interviewees selected their datasets along with their labelling distribution, their input hyperparameters, the model architectures and the achieved metrics as the main points of differentiation. This means the system must consider these as the main points of the configuration of a model training.
- According to most interviewees, new versions of AI models are stored within a cloud configuration management platform, and registered under a new release. On the other side, datasets are stored in a cloud storage platform, separated only by name and with no version tracking, meaning multiple versions of a dataset may coexist within the platform, each one occupying its own space. The target system must solve this issue so as to make the storage of datasets more efficient and scalable.
- Each time the pipeline of a model training completes successfully, some internal software component of the company automatically generates a PDF report with the main information of the training: metrics, performance and the differences with the base model. It would be interesting for the target system to enable the compilation of these reports when model trainings are performed. However, this is not true for all the pipelining software tools used within the target company, to the inclusion of this report should be optional.
- A particular subject admitted to formerly use the traditional method for storing the configuration of models and datasets, but many inconveniences made them take the decision to change to more specialized configuration management systems, such as DVC or WandB. These are technologies mentioned within Chapter 3, and the adoption of a system that integrates these will be easier for some workers.
- Whilst there are still subjects that denied having much issue when spotting a certain model, all of the interviewed workers assure that downloading datasets is troublesome, either due to the download mechanisms of the cloud storage, the location of a dataset within the local file system, the internal company's storage device or the cloud platform, or even by the long download time.
- Many individuals reported that a checkout system would make the search for a dataset quite more easy and time-efficient.
- Some individuals made special emphasis on their need to be able to represent the purpose of the model in some way.
- While there are mixed opinions on the inclusion of a parameter tweaking mechanism, the number of subjects declaring this feature out of the scope of the project outnumbers those who want it within the system.
- Individuals shown particular interest in the inclusion of an *experiment tracking* mechanism within the system, specifically one that can sort the metrics obtained from trainings so that users could spot the best-performing model at a glance.

After compiling all these important points, a software requirement specification was redacted, being the resulting compilation the one shown in *table 4.1*.

Table 4.1: List of functional requirements of the project (previous to organization and priorization).

Requirement ID	Requirement Description
FR1	The system MUST keep ordered track of dataset configurations.
FR1.1	The system MUST detect changes in the size (rows and columns) of datasets.
FR1.2	The system MUST keep track of the routes where models and datasets are stored
FR2	The system MUST handle the concept of experiments.
FR2.1	The system MUST identify an experiment by its identification string, or ID.
FR2.2	The system MUST provide a mechanism for creating experiments.
FR2.3	The system MUST not Accept an experiment creation request that has no experiment name.
FR2.4	The system MUST separate commits from different experiments.
FR2.5	The system MUST provide a mechanism for deleting experiments.
FR2.6	The system MUST provide a mechanism for switching between experiments.
FR3	The system MUST keep ordered track of machine learning model configurations.
FR3.1	The system MUST track which dataset was used to train the model.
FR3.2	The system MUST track which dataset was used to validate the model.
FR3.3	The system MUST track which dataset was used to test the model.
FR3.4	The system MUST track the code used for programming the model's training.
FR3.5	The system MUST track the configuration of the model's hyperparameters and random parameters used in the training of an AI model.
FR3.6	The system MUST track the metrics achieved by an AI model.
FR3.7	The system MUST provide users with a mechanism to mark an AI model as currently deployed.
FR3.8	The system MUST keep track of the additional parameters used by an AI model if it handles domain-specific data. This will be tested for the domain of Earth Observation (EO).
FR4	The system MUST provide a mechanism for evaluating an AI model's metrics' goodness.
FR4.1	The system MUST provide a mechanism for choosing policies to evaluate an AI model's metrics' goodness.
FR4.2	The system MUST provide a mechanism for identifying the experiment or model with the best values on the chosen metrics.
FR4.3	The system SHOULD order the model trainings made on an experiment by the goodness of the values on the chosen metrics.
FR5	The system MUST provide a mechanism for visualizing the performance report for a trained AI model (if any).
FR6	The system MUST provide a mechanism to visualize the training graph of a model (this is, the graph that describes the progress of the metrics of a model after each epoch).
FR7	The system MUST provide users with a mechanism to commit changes to the configuration of an item (both a dataset and an AI model).

Additionally, a list of non-functional requirements was created, based on the needs of the company, these are listed under *table 4.2*.

Table 4.2: List of non-functional requirements of the project.

Requirement ID	Requirement Description
NFR1	The system MUST be modular by design.
NFR1.1	The system MUST be extensible.
NFR1.2	The system MUST be reusable.
NFR2	The system MUST be maintainable.
NFR3	The system MUST be testable.
NFR4	The system SHOULD be power efficient.
NFR5	The system MUST be scalable.
NFR6	The system MUST maintain transactional integrity.
NFR7	The system MUST be fault tolerant.
NFR8	The system MUST be flexible.
NFR9	The system MUST follow best practices for security.
NFR10	The system MUST be usable.
NFR11	The system MUST outperform the traditional method for model tracking and downloading (<2h).

4.2.3. Requirement organisation and prioritization

Since this complex project aims to complete an extensive list of requirements, the development team along with the visionary and the technical coordinator decided in a facilitated workshop held at January 15th,2025, that these requirements shall be reorganized and rearranged by the final specific objective to be completed. That means the project should be divided into milestones. After performing some analysis, three main milestones were defined for the project to succeed:

1. **Milestone 1 (code *D*):** Development of Dataset Configuration Management features completed.
2. **Milestone 2 (code *M*):** Development of Model Configuration Management features completed.
3. **Milestone 3 (code *G*):** Development of General Configuration Management features completed.

Given these milestone definitions, the requirement specification was updated so that every functional requirement was classified into one of the milestones. The new rearranged requirements would be numbered and identified by the milestone they contribute to, resulting into the specifications shown in *table 4.3*, *table 4.4* and *table 4.5*.

Additionally, these tables incorporate the priorities following the procedures described in *section 4.1.2.1*.

Table 4.3: List of functional requirements belonging to milestone 1.

Requirement ID	Requirement Description	priority (MoSCoW)
DFR1	The system MUST keep ordered track of dataset configurations.	Must have
DFR1.1	The system MUST detect changes in the size (rows and columns) of datasets.	Must have
DFR1.2	The system MUST keep track of the routes where models and datasets are stored	Must have

Table 4.4: List of functional requirements belonging to milestone 2.

Requirement ID	Requirement Description	priority (MoSCoW)
MFR1	The system MUST handle the concept of experiments.	Must have
MFR1.1	The system MUST identify an experiment by its identification string, or ID.	Must have
MFR1.2	The system MUST provide a mechanism for creating experiments.	Must have
MFR1.3	The system MUST not accept an experiment creation request that has no experiment name.	Must have
MFR1.4	The system MUST separate commits from different experiments.	Must have
MFR1.5	The system MUST provide a mechanism for deleting experiments.	Must have
MFR1.6	The system MUST provide a mechanism for switching between experiments.	Must have
MFR2	The system MUST keep ordered track of machine learning model configurations.	Must have
MFR2.1	The system MUST track which dataset was used to train the model.	Must have
MFR2.2	The system MUST track which dataset was used to validate the model.	Must have
MFR2.3	The system MUST track which dataset was used to test the model.	Must have
MFR2.4	The system MUST track the code used for programming the model's training.	Must have
MFR2.5	The system MUST track the configuration of the model's hyperparameters and random parameters used in the training of an AI model.	Must have
MFR2.6	The system MUST track the metrics achieved by an AI model.	Must have
MFR2.7	The system MUST provide users with a mechanism to mark an AI model as currently deployed.	Should have
MFR2.8	The system MUST keep track of the additional parameters used by an AI model if it handles domain-specific data. This will be tested for the domain of Earth Observation.	Must have
MFR3	The system MUST provide a mechanism for evaluating an AI model's metrics' goodness.	Should have
MFR3.1	The system MUST provide a mechanism for choosing policies to evaluate an AI model's metrics' goodness.	Should have
MFR3.2	The system MUST provide a mechanism for identifying the experiment or model with the best values on the chosen metrics.	Should have

Continuing on the next page

Table 4.4

Requirement ID	Requirement Description	priority (MoSCoW)
MFR3.3	The system SHOULD order the model trainings made on an experiment by the goodness of the values on the chosen metrics.	Should have
MFR4	The system MUST provide a mechanism for visualizing the performance report for a trained AI model (if any).	Should have
MFR5	The system MUST provide a mechanism to visualize the training graph of a model (this is, the graph that describes the progress of the metrics of a model after each epoch).	Could have

Table 4.5: List of functional requirements belonging to milestone 3 (expanded due to space availability).

Requirement ID	Requirement Description	priority (MoSCoW)
GFR1	The system MUST provide users with a mechanism to commit changes to the configuration of an item (both a dataset and an AI model).	Must have
GFR1.1	The system MUST Identify commits by their Identification string, or ID.	Must have
GFR1.2	The system MUST provide a mechanism for creating commits.	Must have
GFR1.3	The system MUST not Accept the creation of a commit that does not have a commit title.	Must have
GFR1.4	The system MUST allow users to leave a commit message when building up a commit.	Must have
GFR1.5	The system MUST provide users with a mechanism to push a commit to a certain remote repository location.	Must have
GFR1.6	The system MUST provide users with a mechanism to go back to a previous commit.	Must have
GFR1.7	The system MUST be able to handle two instances of a same dataset or model, in different versions, within the same repository.	Must have

4.2.4. Iteration planning

According to *section 4.1.2*, prototyping is one of the main techniques within DSDM-based methodologies. Hence, the iterations of the development of the project have been organised according to prototype production. In other words, the output of an iteration of the *Design and development* phase of the project is a project prototype.

Each prototype will be assigned a subset of requirements of a specific milestone, following the milestone definition made at *section 4.2.3*. The prototype (and therefore, the iteration of the project) is considered as completed when all the requirements of the prototype are developed. The development of a prototype may require performing minor changes on the development of previous prototypes. This fact motivates the definition of a versioning protocol for these prototypes, which will be defined

in the following subsection. Right afterwards, the prototypes will be defined and then arranged into the roadmap of the project.

4.2.4.1. Prototype version control protocol

Given the three milestones defined at *section 4.2.3*, the name of a prototype will consist of the code of the milestone it corresponds to, and the iteration number it corresponds to within that same milestone (e.g., the first prototype for Milestone 3 would be called *G1*).

For any addition to an already-developed prototype related with the requirements contained within it (e.g., a hotfix), semantic versioning will be used. For instance, if a hotfix is needed to prototype *G1*, the new version of the prototype will be called *G1.0.1*. For minor versions, the second number will be increased. Of course, a higher major number for a prototype will indicate that it includes the past prototypes for its milestone[21]. If no prototypes have been developed for a particular milestone at a given moment, the iteration number for that milestone will be considered 0.

The name for a version of the system will be the mixture of all the prototypes present up to the point the version is released. As an example, a version of the system that only contains the first prototypes of milestones 1 and 2 would be called *D1M1G0*.

4.2.4.2. List of prototypes of the project

Prototypes for milestone 1

- **Prototype D1:** Provides the necessary mechanisms for the system to be able to integrate and transparently locate a dataset in the system (establishing a versioning protocol).
- **Prototype D2:** Apart from locating a dataset, the system is able to detect simple changes in the datasets, particularly in the number and content of the features and instances contained within them. Additionally, the system will be able to bring a dataset from a specific location to the user.

Prototypes for milestone 2

- **Prototype M1:** this inital prototype is focused on the integration of the concept of *experiments* in the system, being able to create and track multiple experiments, apart from sepparating the progress made among the different experiments.
- **Prototype M2:** once the system is able to differenciate the different experiments, the system must be able to track the different characteristics of a model upon training. This involves tracking the used datasets, the hyperparameters used, the metrics achieved, and experiment data related to the domain of application (in the current case, the domain of Earth Observation).
- **Prototype M3:** the system is able to used the historically collected metrics from an experiment in order to extract key information like, for example, which of the historically performed experiment runs provided the best performance. Additionally, it provides mechanisms for visualizing both the performance report and the training graph produced by the training of an AI model.

Prototypes for milestone 3

- **Prototype G1:** The system provides a mechanism for users to commit changes to an item (a dataset or an AI model) in order to store them as historical versions in the system.

4.2.4.3. Roadmap of the project

On *figure A.1*, the dependencies among iterations (and hence, the dependencies among the prototypes) are shown. The main dependencies encountered are:

1. Prototypes *D2* and *M2* depend both on *G1*, since both of these prototypes generate items that may be of great value to commit. Hence, it would be valuable to already have mechanisms that allow to store the items within a configuration management system. This makes *G1* the prototype on which more iterations depend and, therefore, motivates it to be the first one to be completed.
2. Prototype *M2* depends on prototype *D1*, since the former needs to identify the datasets used upon the training process of a model, and will use whatever versioning system is defined at the latter to reference these datasets.

Considering the purpose of the prototypes and their dependencies, the designed roadmap is shown at *table 4.6*.

Table 4.6: Roadmap for the project. For details about the stages and dependencies, see *figure A.1*.

Iteration	Prototypes and Phases involved
Iteration 1	• Milestone 3 Analysis • Prototype *G1*
Iteration 2	• Milestone 1 Analysis • Prototype *D1*
Iteration 3	• Prototype *D2*
Iteration 4	• Milestone 2 Analysis • Prototype *M1*
Iteration 5	• Prototype *M2*
Iteration 6	• Prototype *M3*

4.2.5. Time estimations for the project

With the clear picture of what prototypes provide more value and have more dependencies, an estimation was made of the investment of the time that will be required to carry out each iteration of the project.

4.2.5.1. Early time estimation

A first estimation was made before the beginning of the *Analysis* phase of the project, and can be seen in *table 4.7*. The prototypes are ordered sequentially according to the proposed development order, prioritizing those iterations whose output prototype constitutes a dependency to others. The higher the number of development order, the further in the timeline the iteration shall be scheduled.

Table 4.7: Time estimations for all the project iterations.

Iteration	Time estimation (Days)
Iteration 1	15
Iteration 2	15
Iteration 3	10
Iteration 4	15
Iteration 5	10
Iteration 6	10
Total	75

4.2.5.2. Actual time duration

During the course of the project, the time spent in each iteration differed from the initial time estimation. The actual time durations are shown in *table 4.8*.

Table 4.8: Actual time durations for all the project iterations.

Iteration	Time duration (Days)
Iteration 1	18
Iteration 2	21
Iteration 3	9
Iteration 4	20
Iteration 5	14
Iteration 6	9
Total	91

The main reason for this 16-day difference is mainly because of prototypes *D1* and *M1*, which required more time to be completed in order for the development team to be familiarized themselves with the used technological tools used, and finding the ways for them to be integrated within the project. The fact that an initial analysis was also included within iterations 2 and 4 is another reason for this difference.

4.3. EMPLOYED EQUIPMENT AND RESOURCES

A key aspect of the description of the project is not only the techniques used to carry out the development, but also the resources (in this case, hardware and software) utilized in the process. Coming on this section and its subsections, the details about the different equipment and resources used within the development of MADTrack.

4.3.1. Hardware resources

Hardware resources constitute the set of physical devices and peripherals used to carry out the development of the project.

4.3.1.1. Computer

During the development of the project, the target company provided the development team with a laptop computer. This computer is the main means through which the main programming were performed, through which the main documentation and this document were written. This laptop was also used during the different facilitation workshops held during the course of the project.

This computer model is HP OMEN 16-b0xxx, running on the specifications shown at *table 4.9*.

Table 4.9: Specifications of the computer used in the development of the project.

Operating System	Ubuntu 24.04 LTS
CPU	Intel Core i5-1035G1 2.00 GHz
CPU Cores	16
External GPU	NVIDIA GeForce RTX™ 3060 Laptop GPU
Integrated GPU	Intel® UHD Graphics (TGL GT1)
RAM Memory	Tiger Lake-H 16GB (2x8GB)
Storage Memory	1TB NVME™ SSD M.2

4.3.2. Software resources

Software is the intangible material and tools within computers, in this case those that are used to develop the project.

4.3.2.1. Programming Languages

The programming languages used for implementing the components of the project have been:

- **Python:** It is the programming language with the most synergies with AI models and libraries, incorporating powerful frameworks like SKLearn, Pytorch or Tensorflow. It also provides simple yet effective libraries and utilities to carry out client-server communications.
- **Makefile:** the programming language used for automatizing tasks like compilation, execution and environment cleaning, offering a simple, flexible and effective tool for the deployment and testing of the project.
- **Bash Scripting:** used for the automatic deployment of the Tracking Server, a component that will be described in more detail within the coming chapters.

4.3.2.2. Specialized software and libraries

Specialized software and libraries enable developers to perform certain particular complex tasks without reinventing software. These are the libraries and specialized software used during the course of the development of the project:

- ***GitPython*:** a Python library that handles operations on Git repositories. It will be used for generating commits in local repositories and initializing them, as well as accessing other Git operations.
- ***DVC*:** an open-source software that is also a Python library that provides mechanisms for handling operations using the DVC tool. DVC is a tool that enables configuration management for big datasets inside a Git repository.
- ***MLFlow*:** an open-source software that is also a Python library. Provides a REST API for tracking and logging machine learning experiments.
- ***Docker*:** a software dedicated to managing containers, which are isolated environments for running applications. It is used for the creation of containers as part of the deployment of the Tracking Server.
- ***Pandas*:** a Python library that provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It is used for retrieving certain fields out of a training configuration.

- ***PyTest*:** a Python library that provides a simple and flexible testing framework. It was used for preparing, running and validating the different unit tests of the components of the project.

4.3.2.3. Interactive Development Environment (IDE)

The only IDE used during the development of the project is *Visual Studio Code*, which is a free and open-source code editor that provides a rich set of utilities for software developers, such as a file navigator, a wide-ranging variety of extensions, and interactive language support for a wide range of programming languages.

4.3.2.4. Versioning control

The versioning control for this project was managed using *Git* and its web storage service, *GitHub*. It is the most used version control tool used for an immense variety of software projects. Taking advantage of the use of these tools and platforms by the end users, it was decided to store this project's code within their environment, as well as make use of it for keeping track of the development state of the project.

As an additional fact, the use of the *fork* mechanism provided by GitHub was particularly useful for developing an integration test between the target system and a generic version of the pipeline used in real projects. Another functionality of GitHub that was vital for the correct versioning of the project was the *release* utility, which allows to create releases for the project and its components.

4.3.2.5. Documentation tools

The documentation of the project (as well as the documentation of the memory of this Final Degree Project) was developed using the following tools:

- **LaTeX:** this programming language is specialized on producing professional documents, such as papers or scientific reports. It is the main tool used for the production of this document.
- **Canva:** a web-based design tool that enables the creation of high-quality infographics. This was used for the creation of most of the project's non-UML diagrams (e.g., the dependency diagram of the stages of the project (*figure A.1*)).
- **Visual Paradigm Online:** another web-based design tool specialized in UML diagrams. It was used specially for the creation of the project's use case diagrams.
- **Drawio:** another web-based design tool specialized in UML diagrams. It was used specially for the creation of the project's components' class diagrams.

CHAPTER 5

General Configuration Management in MADTrack

One of the main milestones presented within the previous chapter is milestone 3 (codename *G*), which is focused on the development of features related to General Configuration Management. The main purpose of developing this features as a separate module is to provide an extensible basis for possible future developments of features unrelated to versioning datasets and models, and more related to basic configuration management mechanisms. In this chapter, a brief introduction to what needs to be developed will be made, as well as an analysis of the requirements that will need to be developed for this milestone, which will result into a diagnosis of the most suitable architecture and application type for this module.

5.1. INTRODUCTION

The partial objectives of the project are related to configuration management applied to datasets and models, but the development of such a system will depend on more generic configuration management mechanisms. In fact, it is shown on the project's roadmap (*figure A.1*) that the development of the general configuration management features is crucial for initiating the development in the rest of the milestones' prototypes. Hence, the development of this module will be focused on creating mechanisms that interact synergically with the rest of the components, mainly in the cases where commits to remote or local repositories are required.

5.2. REQUIREMENTS ANALYSIS FOR MILESTONE 3

The main purpose of analysis is to define the most suitable way to proceed with the development of the prototypes. In this case, the functionalities provided by Git are the are the ones more suitable to use in the context of development. This Prototype interacts with all those prototypes that require to integrate changes to the system, such as tracking and committing changes within a dataset or a model.

5.2.1. Requirements involved

The requirements involved within this milestone have already been listed within *table 4.5*. These requirements all belong to prototype *G1*, which is the first one to be completed, and the only one belonging to this milestone.

Additionally, prototypes within milestone 3 require the satisfaction of the non-functional requirements listed within *table 4.2*.

5.2.2. Integration with the rest of the system

Prototype G1 is the prototype which constitutes the core dependency for the most prototypes to be developed within the other two milestones. The depending prototypes are D2, which needs a mechanism for committing changes made to DVC's files, and M2, which would potentially need a mechanism for committing metadata stored about training configurations.

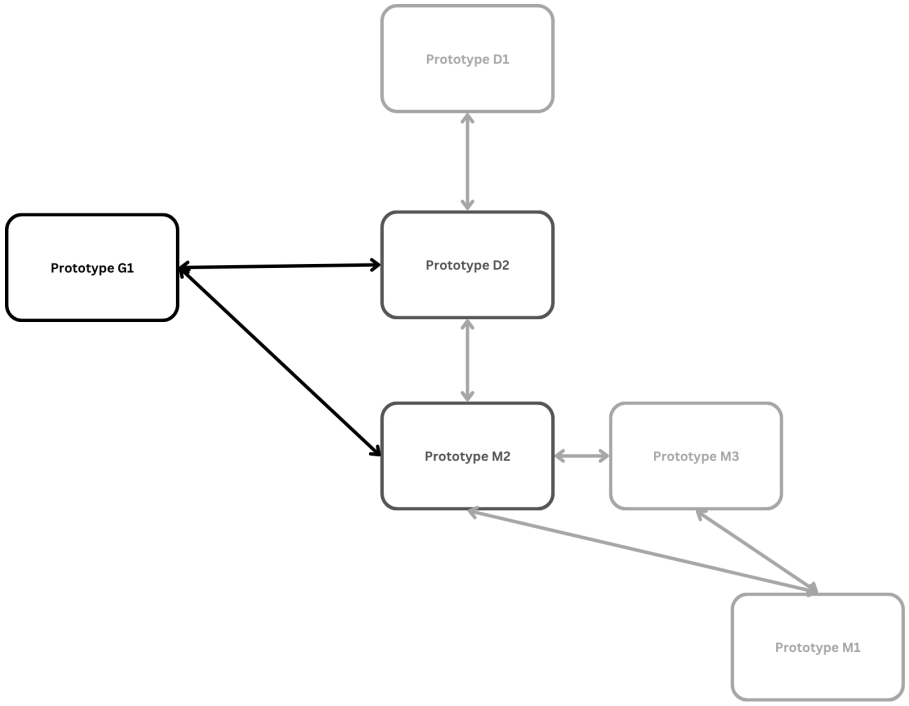

Figure 5.1: Dependencies between prototypes of milestone 3 and the rest of the target system, shown by an interaction diagram.

5.2.3. Architecture analysis

The nature of the requirements of the milestone describe that multiple aspects shall be handled: The user may want to push their changes to either a local or remote repository. And the way to commit changes differs from whether the committed item is a dataset (The information of the DVC tracking file must be committed) or an AI model (whole model folders can be committed from a repository within the remote tracking server). Additionally, as versions would be defined by the commit name, a functionality to retrieve the name of the active commit could be useful.

Considering these needs and the aforementioned requirements, the conclusion reached was that the most suitable architecture for the modules of milestone 3 would mainly consist of a monolithic application, with the possibility for it to connect momentarily to a remote repository server to push changes.

5.2.4. Application type analysis

Due to the reasons described in the previous subsection, the best option for the development of prototypes of this milestone shall be a wrapper library that encapsulates Git operations. A library that allows to commit to local or remote repository independently on the item to be committed, facilitating thus its integration with toolkits that enable the tracking of datasets and models.

5.2.5. Toolkit analysis

Any API or library handling basic Git committing operations will be helpful for the development of the prototypes. Details on the final toolkit used are described in the coming subsections.

5.3. PROTOTYPE DESIGN AND DEVELOPMENT

Within the coming sections, details for the design, implementation and testing phases of the development of this milestone's prototypes are shown in a sequential order. Since this milestone is composed of a single prototype, its development will be described in the following subsection.

5.3.1. Prototype *G1*

Prototype G1 is the only prototype of milestone 3, and the core dependency of many other prototypes. It has the aim of bringing basic Git operations to a library accessible to the end users.

5.3.1.1. Design for prototype *G1*

The use case diagram for prototype *G1* is shown in *figure 5.2*.

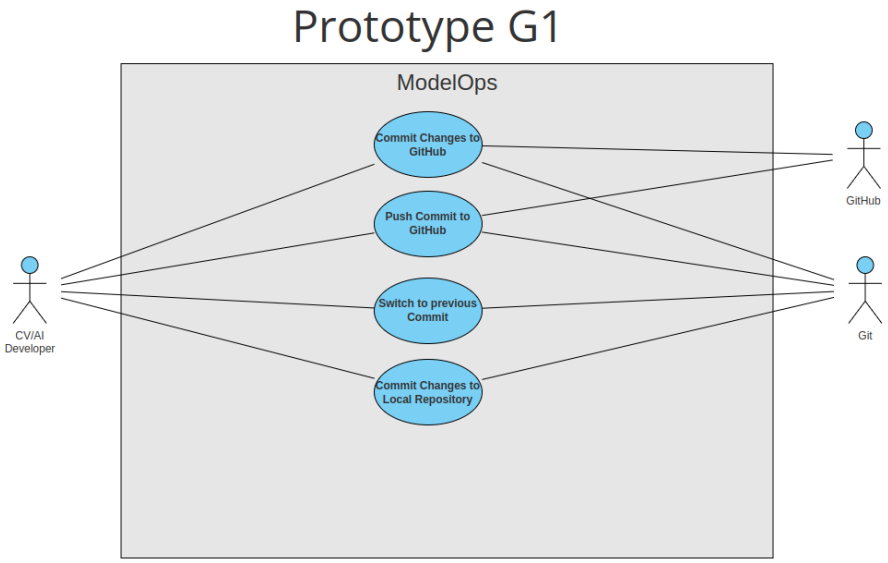

Figure 5.2: Use case diagram for prototype *G1*.

Taking a look at this diagram, it is clear that there is a crucial component needed to carry out this prototype: a component that handles the Git *commit* and *push* operations, along with the possible exceptions that this may generate.

Prototype *G1* components

- **Commit Manager:** a component that will contain the functions for managing both local and remote repositories, as well as operations for chaning the active commit (*checkout*).
- **Commit Exceptions:** A package that includes some special exceptions that can be raised by the Commit Manager.

Design output

The output design class diagram for prototype *G1* is shown in *figure A.2*.

5.3.1.2. Implementation for prototype *G1*

This section addresses questions about the programming language used, the libraries imported and a high-level description of the implementation of the components' methods.

Libraries used

The libraries used for the implementation of this prototype are:

- ***GitPython*:** the selected toolkit to handle *commit*, *push* and *checkout* operations.
- ***Logging*:** A library that enables the creation of log files and manages log operations within any Python application.
- ***OS*:** enables interactions with the operating system, mainly used in this prototype for verifying the existence of particular files and directories.

5.3.1.3. Testing for prototype *G1*

Within the following sections, details for the testing phase of prototype *G1* are shown. This includes the description of the different equivalence classes found for the prototype, and the errors found and lessons learned during the verification of the prototype.

Test suites for Commit Manager

The test suites for this class can be found in *annex reference placeholder*.

Errors found and lessons learned

The errors encountered during the course of the testing phase of prototype *G1* are shown in *annex reference placeholder*.

CHAPTER 6

Dataset configuration management in MADTrack

This chapter will cover the development of the features described in milestone 1 (codename *D*), which is focused on the development of features related to Dataset Configuration Management. This module will work in conjuction with the general configuration management module (see *chapter 5*) to provide mechianisms to track and control the changes of the biggest of files. This chapter will introduce the main issues and address the problems to solve to achieve the goals defined for this milestone.

6.1. INTRODUCTION

One of the partial objectives defined for this Final Degree Project involves the development of a module for tracking and versioning datasets. These datasets can be of variable size, but in many cases bigger than what a free-of-charge configuration management platform can handle. The lack of the existence of a platform that can handle such issue at at a feasible cost motivates the creation of this dataset tracking module. The main focus of the prototypes of milestone 1 is to provide a mechanism for tracking changes within datasets, without the need of committing them to the configuration management platform. Another issue to be tackled is that of bringing the correct dataset to the correct environment, providing at least an interface for this purpose. Firstly, an analysis for the requirements defined for this milestone will be made, separating them by the most suitable prototypes to be developed, to then proceed to the development details of each of these.

6.2. REQUIREMENTS ANALYSIS FOR MILESTONE 1

Analysis has the purpose of defining the high-level outline of the application. The definition of this outline requires addressing the question of the most suitable architecture and application type for the prototypes of this milestone. This analysis will also define the characteristics of a tookit suitable for their development. Furthermore, the protocol for versioning datasets will also be defined within the analysis phase.

6.2.1. Requirements involved

The requirements involved within this milestone have already been listed within *table 4.3*. Since this requirements are contained within different prototypes, it is particularly convenient to define which requirements belong to which prototype. The final requirement division is defined within *table 6.1* and *table 6.2*.

Apart from the final requirement division for each prototype, all prototypes must satisfy the non-functional requirements listed within *table 4.2*.

Requirement ID	Requirement Description	priority (MoSCoW)
DFR1.2	The system MUST keep track of the routes where models and datasets are stored	Must have

Table 6.1: Requirements for prototype *D1*

Requirement ID	Requirement Description	priority (MoSCoW)
DFR1	The system MUST keep ordered track of dataset configurations.	Must have
DFR1.1	The system MUST detect changes in the size (rows and columns) of datasets.	Must have

Table 6.2: Requirements for prototype *D2*

6.2.2. Integration with the rest of the system

Prototypes within milestone 1 only have meaningful interaction with the general configuration management module, mainly because of their dependency from the committing and checkout methods from milestone 3.

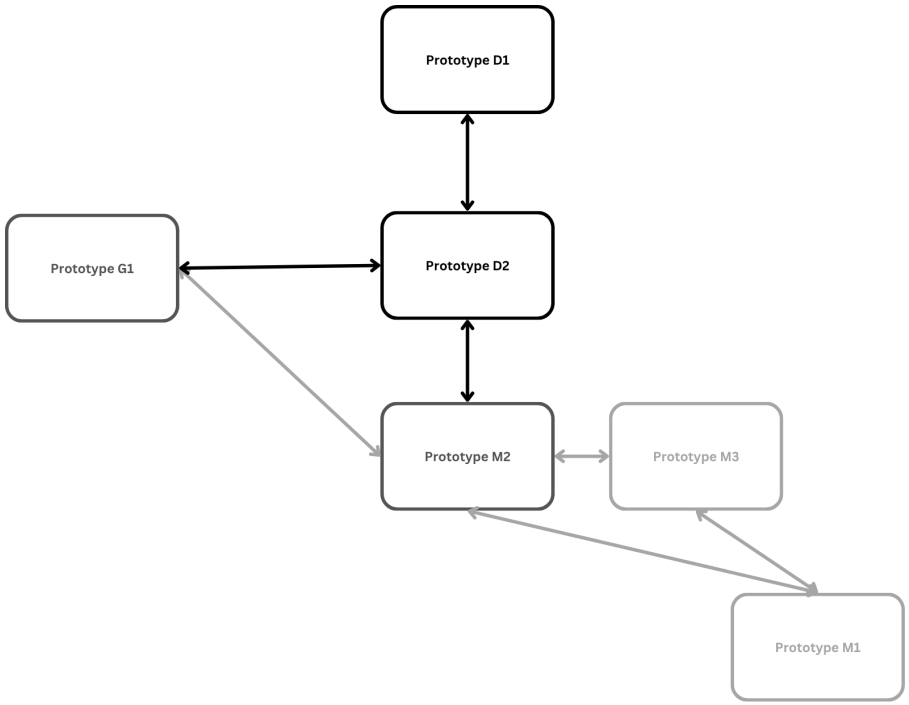

Figure 6.1: Dependencies between prototypes of milestone 1 and the rest of the target system, shown by an interaction diagram.

6.2.3. Architecture analysis

The requirements describe the need for users to be able to locate files regarding datasets (and models, if it were necessary). Hence, an heterogeneous system will be required that is able to load a file or dataset from some sort of initial information, such as an URL. The architecture of this application will be monolithic, but will also have the capability to handle connections to remote file systems.

Also, the coming prototypes will integrate an automatic version detection system that loads the file and sets it in the correct version.

6.2.4. Application type analysis

Since the main objective of this prototype is to locate and load files and perform configuration management operations on them, the most suitable application type is a Wrapper Library that covers online and local operations from a machine with authorized access to the file system.

6.2.5. Toolkit analysis

The most suitable toolkits that may serve for the development of this prototype are those related to dataset configuration management (from *Chapter 3*, the tool with the most potential was DVC), a toolkit for accessing remote file systems securely and locating files within the local machine, and any toolkit providing a mechanism for changing between file versions (the Commit Manager component from prototype *G1* of the system can be used for this purpose).

6.2.6. Versioning protocol for datasets

Any dataset integrated within MADTrack will be versioned using the following naming system:

```
<dataset_name> v<major_version>.<minor_version>
```

where:

- `<dataset_name>` is the name of the dataset.
- `<major_version>` is a positive integer that represents major changes within the datasets without breaking semantics on the purpose of the dataset.
- `<minor_version>` is a positive integer that represents minor changes within the datasets.

6.3. PROTOTYPE DESIGN AND DEVELOPMENT

Within the coming sections, details for the design, implementation and testing phases of the development of this milestone's prototypes are shown in a sequential order. This milestone is composed of two prototypes, whose development details will be shown in the following subsections.

6.3.1. Prototype *D1*

Prototype D1 consists in the initial stage of development of milestone 1. The objective of this prototype is to provide the necessary mechanisms to integrate and transparently locate a dataset within the target system.

6.3.1.1. Design for prototype *D1*

The use case diagram for prototype *D1* is shown in *figure 6.2*.

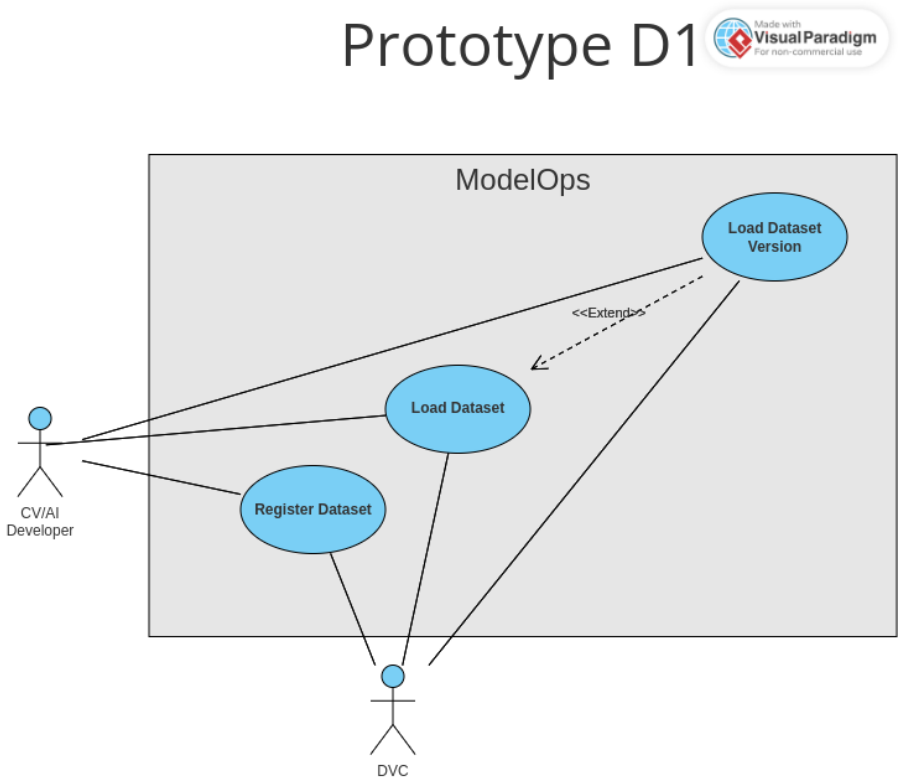

Figure 6.2: Use case diagram for prototype *D1*.

Two main tasks can be extracted from this use case diagram. The first task is related to the tracking of changes within datasets, focusing particularly on enabling this action on a dataset. The second task, additionally, is to enable users to load datasets in either their latest version or any specific existent version. For the fulfillment of this objective, two new components will be created, along with a pack of classes to represent exceptions, errors and specific states for a dataset repository.

Prototype *D1* components

- **Dataset Tracker:** this component will be used for carrying out tasks related with tracking changes on the datasets. For this prototype, it should be capable of starting the change tracking on a dataset and revert its state to a previous, past version.
- **Integration States:** an enumeration component that contains the possible states of integration of a dataset repository into ModelOps' configuration management system: Fully integrated, partially or semi integrated or not integrated.
- **Dataset Fetchers:** a series of components with the main purpose of providing users with the mechanism to obtain datasets from virtually any location, as well as to load them in their different versions, by means of a unified interface to name the dataset versions. For this prototype, only one implementation of this interface will be made.
- **Dataset Exceptions:** a set of exceptions and errors created to represent the possible undesireable or exceptional states that can be encountered by operating the previous two components.

Prototype *D1* design output

The design for prototype *D1* is shown in *Placeholder for annex figure*.

6.3.1.2. Implementation for prototype *D1*

On the coming paragraphs, details for the tools used to implement this prototype are revealed.

Libraries

The libraries used for the implementation of this prototype were:

- ***GitPython*:** this toolkit also helped in this prototype, as the Commit Manager has a synergy with the dataset configuration management module.
- ***DVC*:** this library was used to implement the dataset integration and version change functionalities.
- ***Logging*:** A library that enables the creation of log files and manages log operations within any Python application.
- ***OS*:** this library returns for providing repository existence verification mechanisms.
- ***SHutil*:** a library that enables the creation and removal of directories and files. mainly used for developing functionalities for the Local Dataset Fetcher component.

6.3.1.3. Testing for prototype *D1*

This subsection contains information on the test suites and errors encountered during the testing phase of the prototype.

Test suites for prototype *D1*

The test suites for this class can be found in *annex reference placeholder*.

Errors found and lessons learned during the development of prototype *D1*

The errors encountered during the course of the testing phase of prototype *G1* are shown in *annex reference placeholder*.

6.3.2. Prototype *D2*

Prototype D2 is the second stage of development of Milestone 1. The objective of this prototype is to provide the necessary mechanisms to commit changes to a dataset.

6.3.2.1. Design for prototype *D2*

The output of the design for prototype *D2* is developed by designating the various components that make up this prototype, their contents and their connection with the rest of the system. Hence, the design details for this prototype are shown in the following paragraphs.

Components for prototype *D2*

The use case diagram is shown in *figure 6.3*.

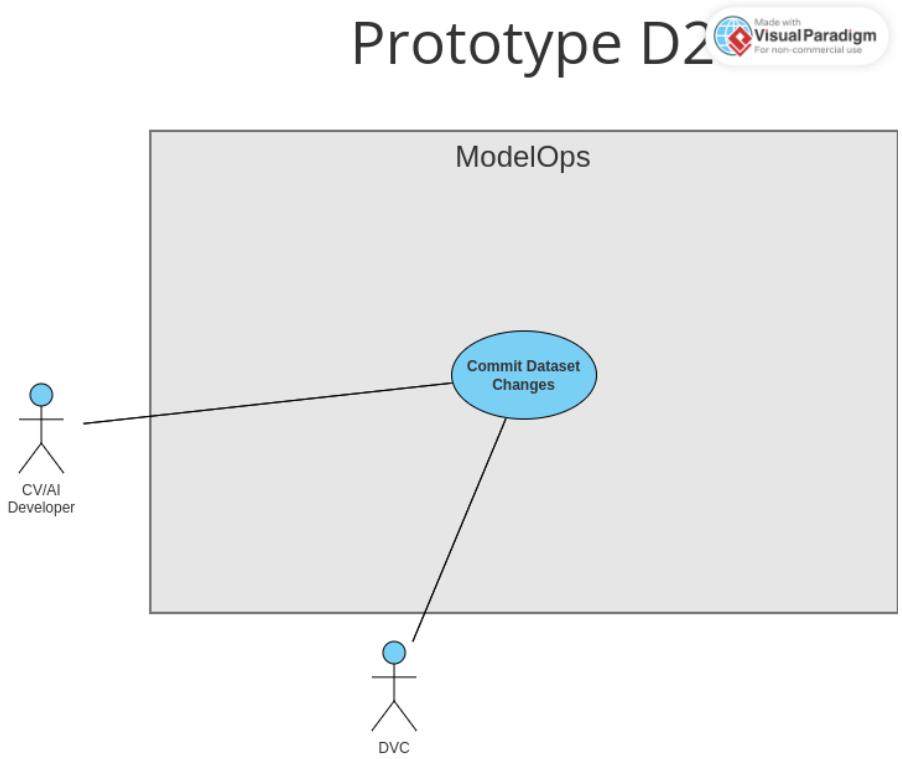

Figure 6.3: Use case diagram for prototype *D2*.

The main task is to perform commits over a dataset. For this objective, the Dataset Tracker component can be used, as it already has a function that allows the user to commit changes to a dataset and generate a new version of it. Hence, the involved components are:

- **Dataset Tracker:** this class will be used for carrying out tasks related with tracking changes on the datasets. For this prototype, it should be capable of creating new versions of a dataset and upload them to a local or remote repository.

Design output for prototype *D2*

The design class diagram for prototype D2 is shown in *Placeholder for annex figure*.

6.3.2.2. Implementation for prototype *D2*

Prototype *D2* mostly follows the same implementation details as the previous prototype of the milestone. Following Python as the main programming language and the same libraries.

6.3.2.3. Testing for prototype *D2*

The testing phase of prototype *D2* was clearly focused on testing the new methods. Since the method `getCurrentDatasetName` is a read-only method that just calls the Commit Manager, its testing can be ommitted. The test suites of the remaining method can be found at *annex reference placeholder* and the errors fixed can be found at *annex reference placeholder*.

Test suites for prototype *D2*

The test suites for this class can be found in *annex reference placeholder*.

Errors found and lessons learned during the development of prototype *D2*

The errors encountered during the course of the testing phase of prototype *D2* are shown in *annex reference placeholder*.

CHAPTER 7

Model Configuration Management in MADTrack

This chapter covers the development of milestone 2 (codename *M*), related to the development of features related to configuration management applied to AI models. This module will provide great value to the end user, increasing the reproducibility of the trainings performed on a given model. For this reasons, this is the chapter with the most prototypes to be developed, and the one with the most complex architecture.

7.1. INTRODUCTION

This chapter aims to provide the necessary documentation of the development of the requirements and features necessary for the fulfillment of one of the partial objectives of the project, which consists in developing a module for versioning AI models. This module will integrate a vital component for deployment, the MADTrack Tracking Server, which will handle the requests from users and store the training configurations. This component, along with the developed solution for the target system that will be used to register the traings, will ensure every single training is accessible for the users, and can be reproduced with similar results in the future.

7.2. REQUIREMENTS ANALYSIS FOR MILESTONE 2

Just as with the other milestones, an analysis of the requirements will be performed to define a high-level outline of what is to come on the following sections. First, the requirements will be grouped semantically according to the milestone 1 prototype that suits best for each requirement, and then make a study of what application type and architecture is more suitable for the module, and what toolkits out of the ones explored on *Chapter 3* would be the most helpful to develop the prototype.

7.2.1. Requirements involved

Just like milestone 1, milestone 2 has a set of requirements that have to be semantically divided into the different prototypes. The requirements belonging to milestone 2 (those with the prefix *MFR*) have been already listed within *table 4.1*, and now they will be divided into three prototypes: *M1* (requirement listed within *table 7.1*), *M2* (requirement listed within *table 7.2*), and *M3* (requirement listed within *table 7.3*).

Requirement ID	Requirement Description	priority (MoSCoW)
MFR1	The system MUST handle the concept of experiments.	Must have
MFR1.1	The system MUST identify an experiment by its identification string, or ID.	Must have
MFR1.2	The system MUST provide a mechanism for creating experiments.	Must have
MFR1.3	The system MUST not accept an experiment creation request that has no experiment name.	Must have
MFR1.4	The system MUST separate commits from different experiments.	Must have
MFR1.5	The system MUST provide a mechanism for deleting experiments.	Must have
MFR1.6	The system MUST provide a mechanism for switching between experiments.	Must have

Table 7.1: Requirements for prototype *M1*.

Requirement ID	Requirement Description	priority (MoSCoW)
MFR2	The system MUST keep ordered track of machine learning model configurations.	Must have
MFR2.1	The system MUST track which dataset was used to train the model.	Must have
MFR2.2	The system MUST track which dataset was used to validate the model.	Must have
MFR2.3	The system MUST track which dataset was used to test the model.	Must have
MFR2.4	The system MUST track the code used for programming the model's training.	Must have
MFR2.5	The system MUST track the configuration of the model's hyperparameters and random parameters used in the training of an AI model.	Must have
MFR2.6	The system MUST track the metrics achieved by an AI model.	Must have
MFR2.7	The system MUST provide users with a mechanism to mark an AI model as currently deployed.	Should have
MFR2.8	The system MUST keep track of the additional parameters used by an AI model if it handles domain-specific data. This will be tested for the domain of Earth Observation.	Must have

Table 7.2: Requirements for prototype *M2*.

Requirement ID	Requirement Description	priority (MoSCoW)
MFR3	The system MUST provide a mechanism for evaluating an AI model's metrics' goodness.	Should have
MFR3.1	The system MUST provide a mechanism for choosing policies to evaluate an AI model's metrics' goodness.	Should have
MFR3.2	The system MUST provide a mechanism for identifying the experiment or model with the best values on the chosen metrics.	Should have
MFR3.3	The system SHOULD order the model trainings made on an experiment by the goodness of the values on the chosen metrics.	Should have
MFR4	The system MUST provide a mechanism for visualizing the performance report for a trained AI model (if any).	Should have
MFR5	The system MUST provide a mechanism to visualize the training graph of a model (this is, the graph that describes the progress of the metrics of a model after each epoch).	Could have

Table 7.3: Requirements for prototype *M3*.

7.2.2. Integration with the rest of the system

Prototypes within milestone 2 have a strong relation with the rest of milestones, since they need the configuration management mechanisms to commit changes made to the folders that track the model training configurations, and will also use the dataset versioning system on dataset logging tasks.

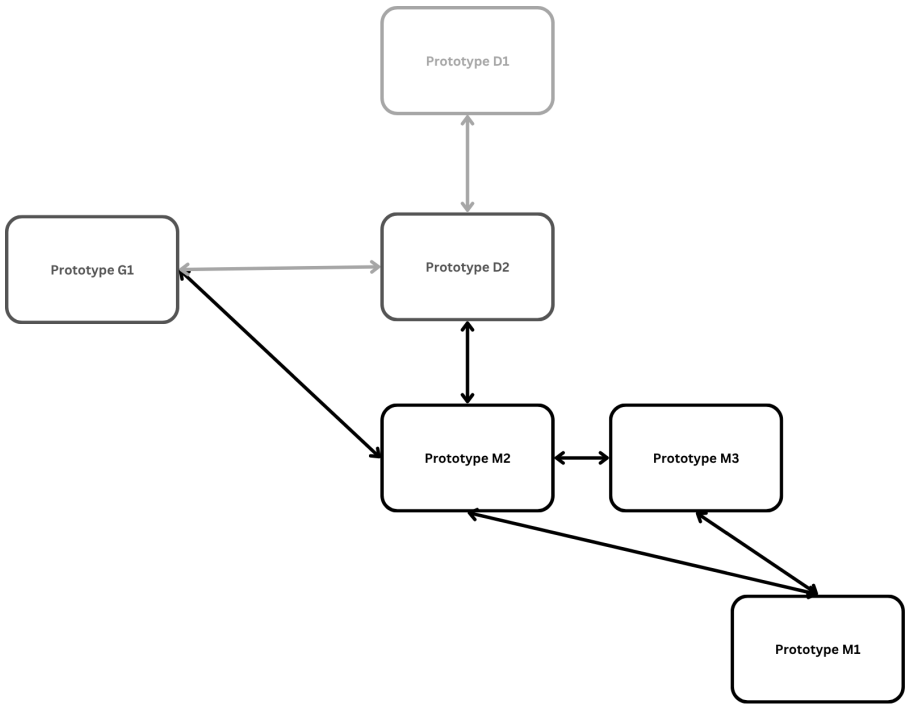

Figure 7.1: Dependencies between prototypes of milestone 2 and the rest of the target system, shown by an interaction diagram.

7.2.3. Architecture analysis

The main objective of the milestone is to enable the target system to handle experiments, model trainings, register and version them. The need for registration of events happening on a machine at the time they are happening arise the need of a component that can manage all of them. This component shall be separate from the rest of the system so as to increase portability. Hence, the architecture of this module will be distributed, with two main components.

7.2.4. Application type analysis

The aim of this prototype is to manage the existence of experiments. Since this is a double-sided task, the best approach is to provide the management functionality inside a Wrapper Library, from which the user will be able to create, delete and switch their active experiment, as well as managing their model trainings using code, whilst a remote tracking server will be in charge of processing these requests with a web application, provided by an external toolkit.

7.2.5. Toolkit analysis

The most suitable toolkit for the development of this milestone is the one that provides a mechanism for creating experiments under the definition of a series of AI model lifecycles performed with the purpose to achieve a suitable model for a specific goal. A nice example is MLFlow (overviewed in *Chapter 3*), which is a tool for model registration and experiment tracking.

7.3. PROTOTYPE DESIGN AND DEVELOPMENT

The following subsections contain details about the design, implementation and teting process for the three prototypes this milestone is composed of.

7.3.1. Prototype *M1*

Prototype *M1* is the first stage of development of Milestone 2 (consisting in AI model versioning and tracking). The objective of this prototype is to provide the necessary mechanisms to register experiments into a remote server, based on their purpose.

7.3.1.1. Design for prototype *M1*

The following sections describe the design process of prototype *M1*.

Components for prototype *M1*

The use case diagram which summarizes this prototype's needs is shown in *figure 7.2*.

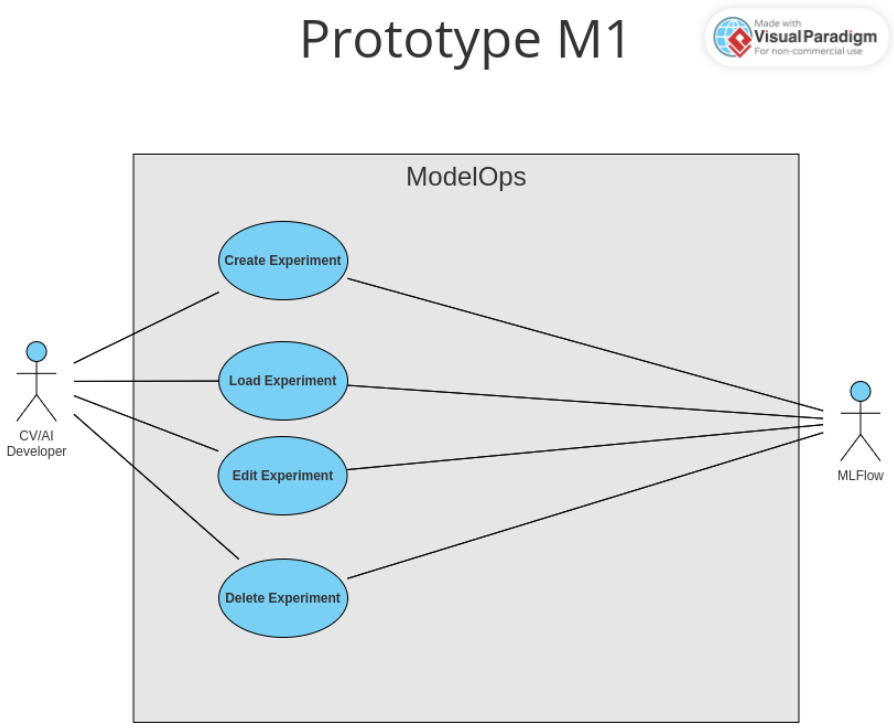

Figure 7.2: Use case diagram for prototype *M1*.

The objective of this prototype is to handle the life cycle of experiments: creation, switching, edition and deletion. For this purpose, a new class with the role of managing this life cycle will be created, along with a new component which will relate to deployment:

- **Experiment Manager:** this class will yield the role of managing the life cycle of experiments, covering all the aforementioned operations. It will be able to create, switch, edit and delete experiments.
- **Tracking Server:** a Distributed component consisting on a docker container, running locally for the meantime, which will contain the Active MADTrack Tracking server which will use a file system as storage.
- **Model Tracking Exceptions** : a package that includes some special exceptions that can be raised by the Experiment Manager.

Design output for prototype *M1*

The design class diagram for prototype *M1* is shown in *Placeholder for annex figure.*

7.3.1.2. Implementation for prototype *M1*

The following lines describe the details of the implementation of the prototype.

Libraries

The libraries used for this implementation are:

- **MLFlow:** an open-source Python library that provides a unified API for tracking and logging machine learning experiments.

- **Logging:** a library that enables the creation of log files and manages log operations within any Python application.
- **OS:** a Python library enabling operating system interaction. It will be used to build the necessary paths to local file repositories.
- **YAML:** A Python library that provides a way to represent data in YAML format using simple structures.

7.3.1.3. Testing for prototype *M1*

The following paragraphs contain information about the test suites and errors fixed during the testing process of prototype *M1*.

Test suites for Experiment Manager

The test suites for this prototypes can be found in *annex reference placeholder*.

Errors found and lessons learned during the development of prototype *M1*

The errors encountered during the course of the testing phase of prototype *M1* are shown in *annex reference placeholder*.

7.3.2. Prototype *M2*

Prototype *M2* is the second stage of development of milestone 2. This prototype will focus on the effective AI logging during the model lifecycle. This objective can be accomplished by providing mechanisms for logging all the datasets (test, train, validation), the code used to train the model, the hyperparameters, and domain-specific data.

7.3.2.1. Design for prototype *M2*

The following sections describe the design process of prototype *M2*.

Components for prototype *M2*

First, it is necessary to look at the use case diagram, (shown at *figure 7.3*), which summarizes this prototype's needs.

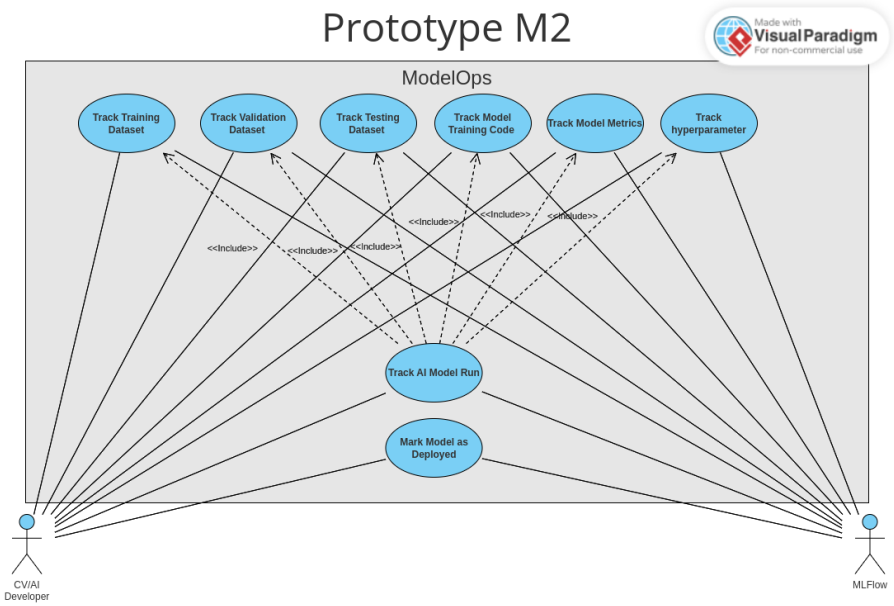

Figure 7.3: Use case diagram for prototype *M2*.

The objective of this prototype is to enable users to fill their experiments with *Model Runs* (for more information about these, please refer to *section 3.4.3*), which will have several data logged according to the needs of the user. For this objective, a new component will be created to represent an AI model run, along with changes in the Experiment Manager component. The list of components involved in this prototype is:

- **Model Run:** This component will be in charge of representing a run of a single model within the training workflow. Its main functionality resides in the component's capability of logging the necessary items to store the configuration of a run.
- **Experiment Manager:** This class will enjoy new functionalities, such as the capability to run a complete run workflow or to register a new AI model version within the model registry.
- **Tracking Server:** This component will be able to store the necessary data to enable logging in the model run. The details about its architecture are revealed in *Chapter*

Design output for prototype *M2*

The design class diagram for prototype M2 is shown in *Placeholder for annex figure.*

7.3.2.2. Implementation for prototype *M2*

This are the lines that contain the main details about this prototypes implementation. The main libraries do not differ from the previous prototype.

7.3.2.3. Testing for prototype *M2*

The particularity of the testing of this prototype is that it could be mostly done from the Experiment Manager component, since it has a method that subcalls the methods from the Model Run component. The test suites of the Experiment Manager will also test the Model Run component.

Test suites for prototype *M2*

The test suites for this class can be found in *annex reference placeholder*.

Errors found and lessons learned

The errors encountered during the course of the testing phase of prototype *M2* are shown in *annex reference placeholder*.

7.3.3. Prototype *M3*

Prototype M3 is the third stage of development of Milestone M (AI model versioning and tracking). This prototype will be focused on how metrics are shown in the MADTrack system and how they are presented to end users, as well as the final presentation of these metrics on a performance report (Which would be logged as another parameter and printed out when completing a Model Run, if necessary).

7.3.3.1. Design for prototype *M3*

The following sections describe the design process of prototype *M3*.

Components for prototype *M3*

The use case diagram is shown in *figure 7.4*, summarizing thus the needs of this prototype.

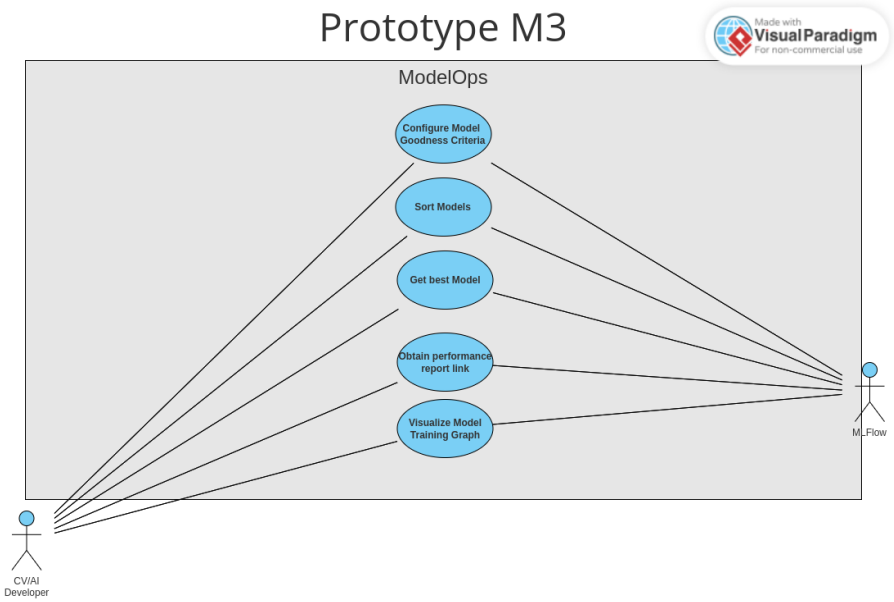

Figure 7.4: Use case diagram for prototype *M3*.

The objective of this prototype is to enable users to extend the functionality of a model run, so that the goodness of the metrics can be configurable, so that the model runs can be sorted according to a specific metric, and so that the best model run can be highlighted in some way. Additionally, the system will also provide a mechanism to log a performance report for the model, which will be

stored as an artifact of the model run and its access will also be provided in the logs, and to visualize the training graph of the model, if it is trained.

For this reason, the main component involved within this prototype is the Model Run component, which will gain additional methods, and the Experiment Manager component.

Design output for prototype *M3*

The design class diagram for prototype M3 is shown in *Placeholder for annex figure*.

7.3.3.2. Implementation for prototype *M3*

The following sections describe the implementation process of prototype *M3*. The language and the libraries used are the same as the rest of prototypes of this milestone.

7.3.3.3. Testing for prototype *M3*

The main testing methods focus on the new methods of the Model Run component.

Test suites for prototype *M3*

The test suites for this class can be found in *annex reference placeholder*.

Errors found and lessons learned during the development of prototype *M3*

The errors encountered during the course of the testing phase of prototype *M3* are shown in *annex reference placeholder*.

CHAPTER 8

MADTrack Tracking Server Deployment

CHAPTER 9

Results and tests

CHAPTER 10
Conclusiones

10.1. OBJETIVOS ALCANZADOS

En este capítulo se realizará un juicio crítico y discusión sobre el objetivo general y objetivos secundarios alcanzados durante el desarrollo del trabajo.

10.2. JUSTIFICACIÓN DE COMPETENCIAS ADQUIRIDAS

Es muy importante recordar que según la normativa vigente en la ESI-UCLM, el capítulo de conclusiones debe incluir *obligatoriamente* un apartado destinado a justificar la aplicación en el TFG de las competencias específicas (una o más) adquiridas en la tecnología específica cursada, como se indica a continuación:

> En el TFG se han aplicado las competencias correspondientes a la Tecnología Específica de *[poner lo que corresponda]*:
>
> **Código de la competencia 1**: *[Texto de la competencia 1]*. Explicación de cómo se ha aplicado en el TFG.
>
> . . . (otras más si las hubiera).

10.3. TRABAJOS DERIVADOS Y FUTUROS

Si es pertinente se puede incluir información sobre trabajos derivados como publicaciones o ponencias en preparación, así como trabajos futuros *(solo si estos están iniciados o planificados en el momento que se redacta el texto)*.

Se recomienda reflexionar sobre la conveniencia de inclusión de una lista de posibles mejoras, ya que puede transmitir la impresión de que el trabajo se encuentra en un estado incompleto o inacabado.

10.4. VALORACIÓN PERSONAL

En esta sección final se realizará un rápido análisis de las lecciones aprendidas en las que se pueden incluir tanto buenas prácticas adquiridas (tecnológicas y procedimentales) como cualquier otro aspecto de interés. También se puede resumir cuantitativamente el tiempo y esfuerzo dedicados al proyecto a lo largo de su desarrollo.

Bibliografía

[1] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Airfocus. Moscow prioritization. URL: https://airfocus.com/glossary/what-is-moscow-prioritization.

[4] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[5] Anna. A comprehensive guide to mlflow: What it is, its pros and cons, and how to use it in your python projects, 2024.

[6] Jasmin Praful Bharadiya, Reji Kurien Thomas, and Farhan Ahmed. Rise of artificial intelligence in business and industry. *Journal of Engineering Research and Reports*, 25(3):85–103, 2023.

[7] Walter R Bischofberger and Gustav Pomberger. *Prototyping-oriented software development: Concepts and tools*. Springer Science & Business Media, 2012.

[8] Alexánder Borbón and Walter Mora. *Edición de textos científicos con LaTeX. Composición, diseño editorial, gráficos, Inkscape, Tikz y presentaciones Beamer*. Instituto Tecnológico de Costa Rica, 2 edition, 2021.

[9] Cyril Cappi, Camille Chapdelaine, Laurent Gardes, Eric Jenn, Baptiste Lefevre, Sylvaine Picard, and Thomas Soumarmon. Dataset definition standard (dds). *arXiv preprint arXiv:2101.03020*, 2021.

[10] Gerry Coleman and Renaat Verbruggen. A quality software process for rapid application development. *Software Quality Journal*, 7:107–122, 1998.

[11] Nimra E. Crowdbotics blog - data version control explained, 2021.

[12] Wix Engineering. Watch: Introducing mlflow and wix's internal machine learning platform. Medium story available at URL: https://medium.com/wix-engineering/watch-introducing-mlflow-wixs-internal-machine-learning-platform-ad9227d85cac, 2020.

[13] Claudia F. Git lfs: The pocketbook explanation. URL: https://get.assembla.com/blog/git-lfs/, 2024.

[14] Industry Guides. *Guide: SOC 2 Reporting on an Examination of Controls at a Service Organization Relevant to Security, Availability, Processing Integrity, Confidentiality, or Privacy*. 2018.

[15] Anne Mette Jonassen Hass. *Configuration management principles and practice*. Addison-Wesley Professional, 2003.

[16] Shawn Hymel, Colby Banbury, Daniel Situnayake, Alex Elium, Carl Ward, Mat Kelcey, Mathijs Baaijens, Mateusz Majchrzycki, Jenny Plunkett, David Tischler, et al. Edge impulse: An mlops platform for tiny machine learning. *arXiv preprint arXiv:2212.03332*, 2022.

[17] IBM. What is a dataset? URL: https://www.ibm.com/think/topics/dataset, 2024.

[18] Edge Impulse. What is edge impulse? Documentation available at URL: https://docs.edgeimpulse.com/docs/concepts/edge-ai-fundamentals/what-is-edge-impulse.

[19] Microsoft. Integrate mlflow and ray. URL: https://learn.microsoft.com/en-us/azure/databricks/machine-learning/ray/ray-mlflow, 2025.

[20] Wojciech Prażuch. Machine learning tools comparison. URL: https://www.netguru.com/blog/machine-learning-tools-comparison, 2025.

[21] Tom Preston-Werner et al. Semantic versioning 2.0. 0. Available at URL: https://semver.org/, 2013.

[22] David Rijlaarsdam, Tom Hendrix, Pablo T Toledano González, Alberto Velasco-Mata, Léonie Buckley, Juan Puig Miquel, Oriol Aragon Casaled, and Aubrey Dunne. The next era for earth observation spacecraft: An overview of cognisat-6. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.

[23] Servicio de publicaciones UCLM. Propiedad intelectual. Documentos elaborados por el Grupo de Gestión del Conocimiento y Propiedad Intelectual de la Universidad de Castilla-La Mancha. URL: https://e.uclm.es/servicios/doc/?id=UCLMDOCID-12-739. Último acceso: feb. 2024.

[24] The Comet.ml Team. Comet.ml official documentation. Available at URL: https://www.comet.com/docs/v2/, 2025.

[25] The DVC Team. Dvc official documentation. Available at URL: https://dvc.org/doc, 2025.

[26] The MLFlow Team. Mlflow tracking. URL: https://mlflow.org/docs/latest/ml/tracking, 2024.

[27] The MLFlow Team. Mlflow official documentation. Available at URL: https://mlflow.org/docs/latest/ml/, 2025.

[28] The Neptune.ai Team. Neptune.ai official documentation. Available at URL: https://docs.neptune.ai/, 2025.

[29] Universidad de Cantabria. Cómo usar imágenes en trabajos. Artículo técnico disponible en URL: https://web.unican.es/buc/Documents/Formacion/guia_imagenes.pdf, 2018. Último acceso: sep. 2021.

[30] G2 users. Comet.ml product reviews. URL: https://www.g2.com/products/comet-ml/reviews, 2025.

[31] G2 users. Neptune product reviews. URL: https://www.g2.com/products/neptune-ai/reviews?utf8=%E2%9C%93&filters%5Bsentiment_snippet%5D=516376#reviews, 2025.

[32] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

ANEXOS

ANEXO A

Annex: Big Figures

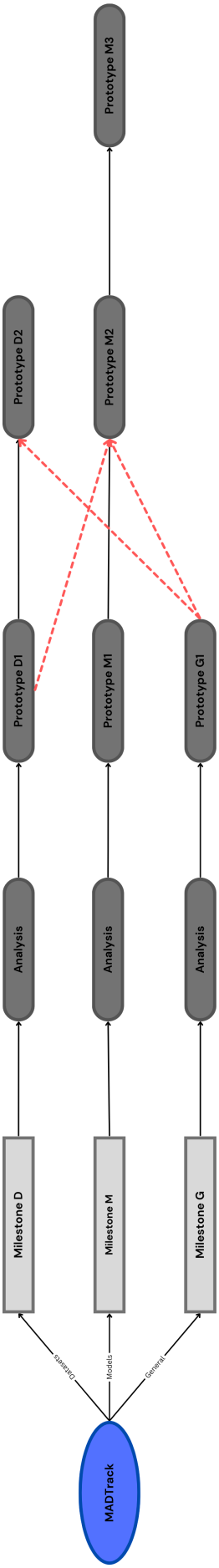

Figure A.1: Iteration roadmap of the MADTrack project.

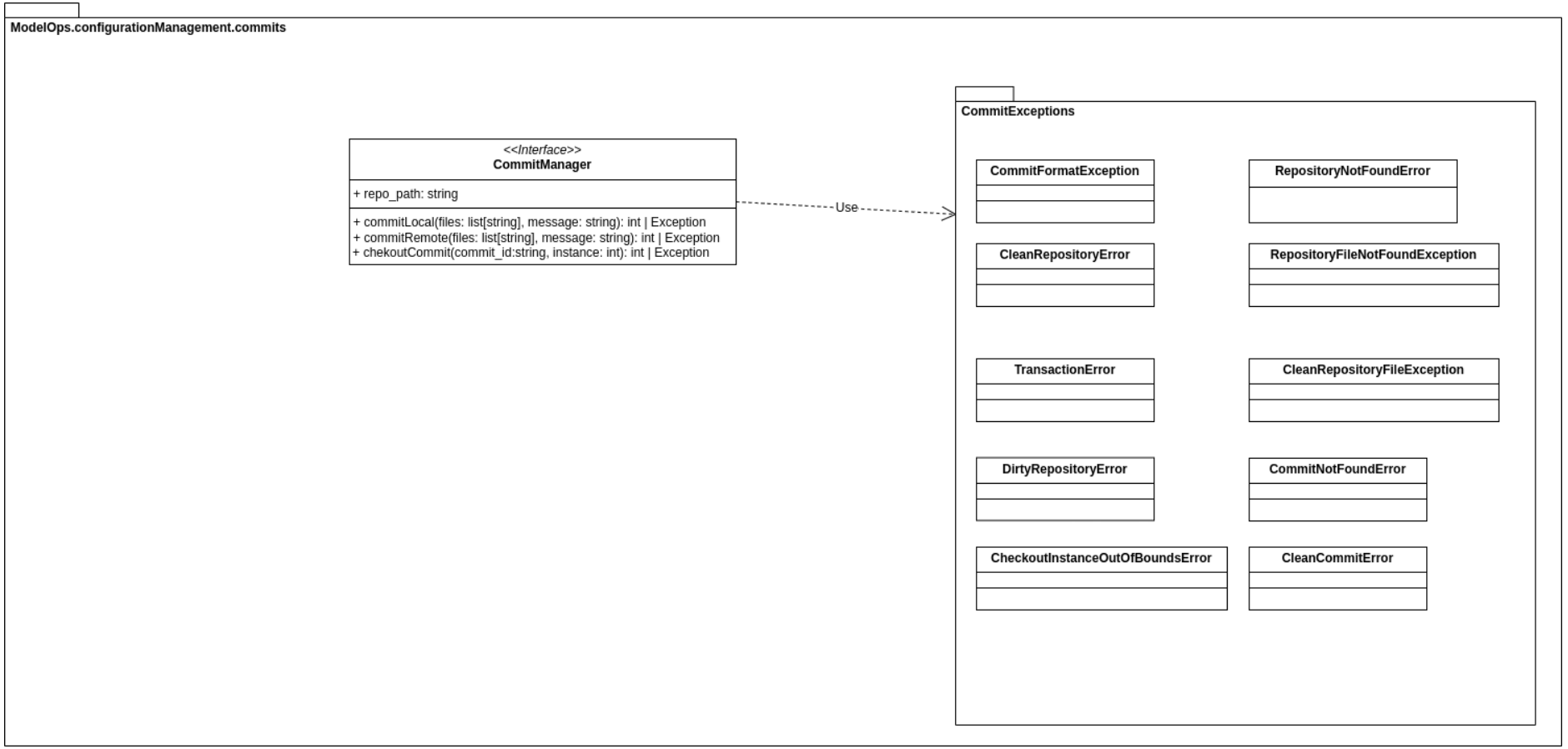

Figure A.2: Class diagram for prototype *G1*.

ANEXO B

Breve introducción a LaTeX

El contenido del trabajo final de estudios se organiza en capítulos que se subdividen en secciones. Con LaTeX este tipo de organización se realiza de modo inmediato mediante la generación automática de los estilos correspondientes a los títulos de cada sección y su inclusión en la tabla de contenidos. Los ajustes relativos a la generación del formato y estilos asociados a secciones del documento se realizan con el paquete `titlesec` empleado en esta plantilla.

En las secciones siguientes se comenta la inclusión con LaTeX de distintos elementos de organización de información junto a ejemplos que facilitan su utilización en la memoria del trabajo.[1]

B.1. LISTAS

Existen dos tipos de listas: enumeraciones y listas con viñetas. En el primer tipo, los elementos de la lista se preceden de una clave numérica o alfabética, mientras que en el segundo tipo se emplea una viñeta. En ambos casos los elementos se pueden anidar para crear una jerarquía entre ellos. En LaTeX se recomienda la inclusión del paquete `enumitem` (incluido en esta plantilla) que permite personalizar fácilmente las listas de un documento. A continuación se muestran algunos ejemplos de las posibilidades.

B.1.1. Ejemplo de lista con viñetas personalizadas

- pera

☞ manzana

🎗 naranja

B.1.2. Ejemplo de lista condensada

Este tipo de lista presenta una separación mínima entre ítems, en varias columnas y configuración de la etiqueta.

(1) pera
(2) manzana
(3) naranja
(4) patata
(5) calabaza
(6) fresa

Además del texto, los documentos pueden incluir elementos que enriquecen su contenido facilitando su exposición y comprensión. En las secciones siguientes tratamos brevemente dichos elementos.

[1] Las explicaciones de este anexo forman parte del contenido del curso «LaTeX esencial para preparación de TFG y otros documentos académicos» de la ESI-UCLM.

B.2. TABLAS

A continuación se incluyen algunos ejemplos de tablas elaboradas con LaTeX mediante el empleo de paquetes dedicados. Para la realización de tablas más complejas se recomienda la consulta de manuales [8] y el empleo de asistentes o herramientas en línea.[2]

Observa que el título de las tablas se ubica en la parte superior de la tabla. Puesto que el contenido de la tabla es texto, tiene sentido leer primero el título para contextualizar el contenido de la tabla antes de su lectura.

Table B.1: Ejemplo de uso de la macro `cline`

7C0	hexadecimal
3700	octal
11111000000	binario
1984	decimal

Ejemplo de tabla en la que se controla el ancho de la celda.

Table B.2: Ejemplo de tabla con especificación de anchura de columna

Día	Temp Mín (°C)	Temp Máx (°C)	Previsión
Lunes	11	22	Día claro y muy soleado. Sin embargo, la brisa de la tarde puede hacer que las temperaturas desciendan
Martes	9	19	Nuboso con chubascos en muchas regiones. En Cataluña claro con posibilidad de bancos nubosos al norte de la región
Miércoles	10	21	La lluvia continuará por la mañana, pero las condiciones climáticas mejorarán considerablemente por la tarde

B.3. ECUACIONES MATEMÁTICAS

La composición de ecuaciones requiere el uso de comandos especializados. Por tanto, para facilitar dicha tarea se aconseja el empleo de programas especializados como MathType o asistentes como el incluido en editores como TeXstudio[3] o herramientas en línea.[4] Es muy simple incluir fórmulas matemáticas sencillas en el mismo texto en el que se escribe. Por ejemplo, $h^2 = a^2 + b^2$ que podría ser la ecuación representativa del teorema de Pitágoras (ver también ec. B.1).

[2] https://www.tablesgenerator.com/
[3] https://www.texstudio.org/
[4] https://latex.codecogs.com/, http://www.sciweavers.org/free-online-latex-equation-editor

Las fórmulas también se pueden separar del texto para que aparezcan destacadas, así:

$$c^2 = \int (a^2 + b^2) \cdot dx$$

Pero si se desea, las ecuaciones pueden ser numeradas de forma automática e incluso utilizar referencias cruzadas a ellas:

$$h^2 = b^2 + c^2 \tag{B.1}$$

B.4. FIGURAS

A diferencia de lo que sucede en las tablas, el título de las figuras aparece en la parte inferior de estas. Para la inclusión de las figuras se debe tener en cuenta que su contenido se encuentra en un fichero individual con el formato y resolución apropiados para garantizar la calidad del resultado final.

En esta sección se añaden ejemplos de muestra para la inclusión de figuras simples y otras compuestas de subfiguras mediante el empleo del paquete `subcaption`.

Figure B.1: Fotografía a color (Fuente: J. Salido, CC BY-NC-ND)

Ejemplo de figura compuesta por dos subfiguras incluidas mediante paquete `subcaption`. A través del uso de etiquetas (`\label`) es posible incluir referencias cruzadas a subfiguras como la fotografía en blanco y negro de la Fig. B.2b.

(a) Fotografía a color **(b)** Fotografía en blanco y negro

Figure B.2: Ejemplo de inclusión de subfiguras en un mismo entorno (Fuente: J. Salido, CC BY NC ND)

En los trabajos académicos la inclusión de imágenes y figuras que no son propiedad del autor suscitan bastante controversia, ya que con frecuencia se incumple inadvertidamente la ley vigente de propiedad intelectual. Respecto a este hecho se recomienda, tanto al alumnado como al profesorado encargado de la tutorización, consultar documentación informativa sobre el uso correcto de figuras en documentos académicos [23, 29]. Entre las «incorrecciones» más habituales en los documentos académicos, se observa:

- *Abuso del derecho de cita.* Se produce al incluir, con fines exclusivamente decorativos o ilustrativos de la explicación, una figura sujeta a derechos de uso restringido invocando el derecho de cita (incluso con correcta atribución de la obra).
- *Incorrecta atribución de la obra.* Es habitual confundir al autor de la obra con la fuente de origen de la misma. La fuente es precisa cuando se cita la obra original. Sin embargo, la licencia de muchas obras exige la atribución al autor y la inclusión de la licencia bajo la que se distribuye o hace uso de la misma. Véase como ejemplo cómo se realiza una correcta atribución en las Fig. B.1 y B.2 mencionando al autor y la licencia Creative-Commons[5] bajo la que se rige el uso de la imagen y el mecanismo de título alternativo para que dicha atribución no aparezca en el índice de figuras usando título opcional.
- *Supresión de los detalles de la licencia de uso.* Al incluir obras de terceros debemos tener presente los términos de distribución de la misma e incluirlos junto a la atribución de autoría.

La inclusión de material de *dominio público*, sin restricciones de uso o con permiso, hace innecesaria la atribución al autor, pero se recomienda incluir una nota de agradecimiento.[6]

Cuando se presenta la necesidad de incluir un gráfico demasiado grande para el tamaño de la página, una opción muy apropiada es la impresión del gráfico en modo girado en una página aparte. Este efecto se consigue con el entorno `sidewaysfigure` proporcionado por el paquete `rotating`. La Fig. B.3 muestra un ejemplo del entorno citado con un gráfico PDF.

[5] https://creativecommons.org

[6] Incluyendo un texto como: *«Por cortesía de ...»*

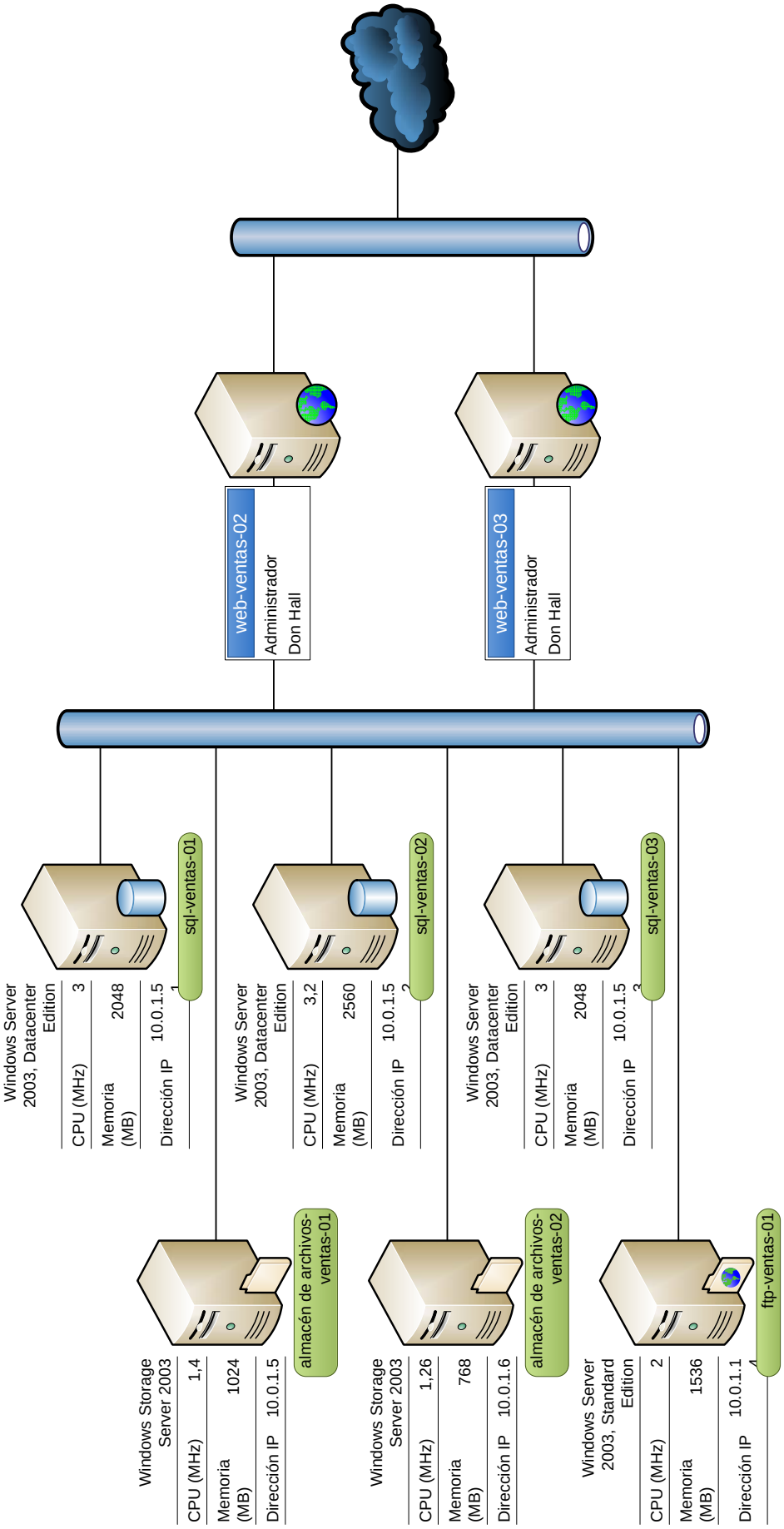

Figure B.3: Figura vectorial con impresión girada

También es posible imprimir una página en formato apaisado cuando contiene una figura muy ancha. Este efecto se consigue con el paquete `pdflscape` y el entorno `landscape` proporcionado. Además, es este caso se han suprimido tanto la cabecera como el pie de página. La figura B.4 se muestra apaisada a modo de ejemplo.

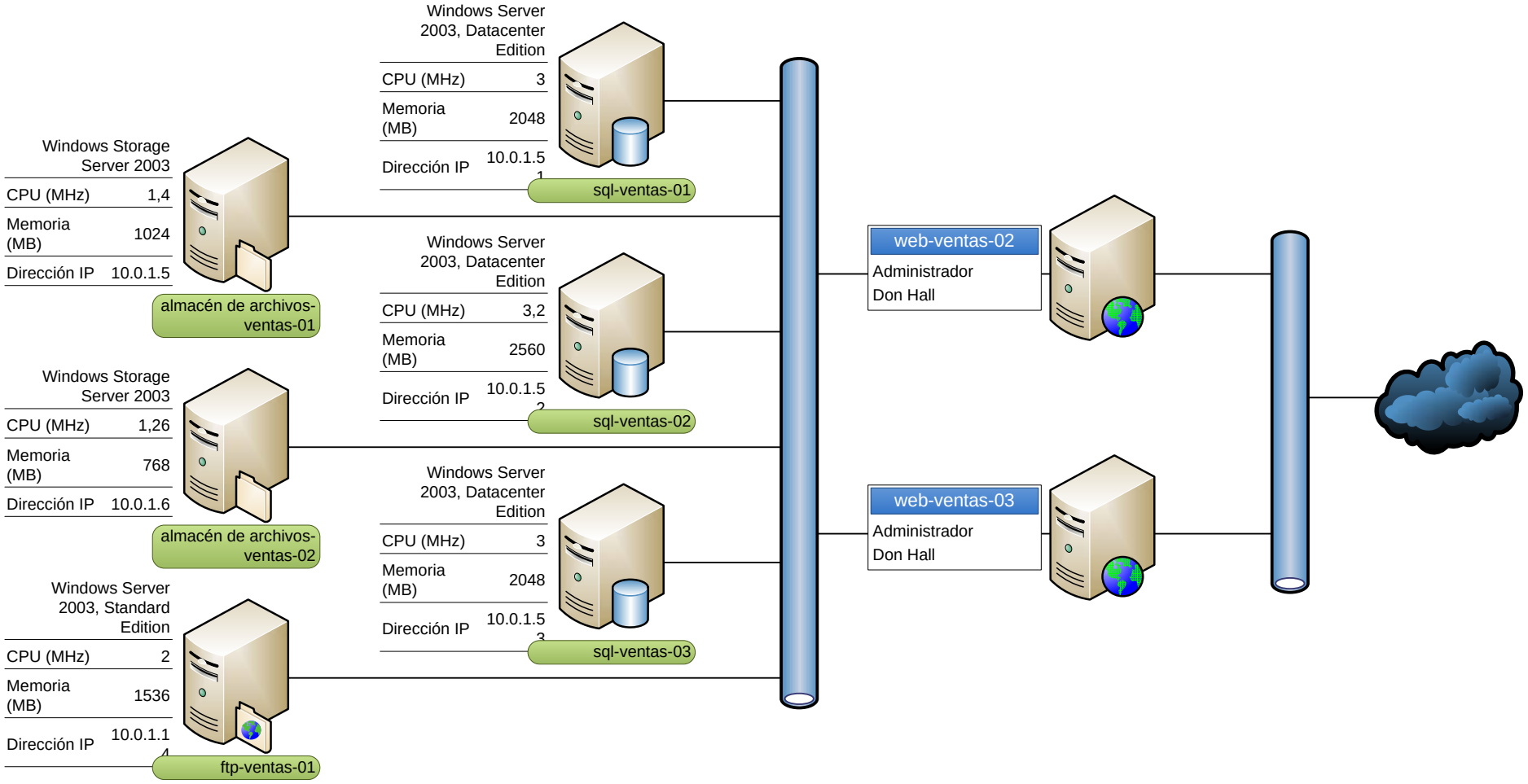

Figure B.4: Figura vectorial con vista en página apaisada

B.5. LISTADOS DE CÓDIGO FUENTE

En los textos científicos relacionados con las TIC[7] (Tecnologías de la Información y Comunicaciones) suelen aparecer porciones de código en las que se explica alguna función o característica relevante del trabajo que se expone.

La inclusión de porciones de código fuente se puede formatear de modo sencillo en LaTeX mediante el uso del paquete `listings`. A continuación, se muestran varios ejemplos de porciones de código correspondientes a distintos lenguajes de programación.

Listado B.1: Ejemplo de código fuente en lenguaje Java

```java
// @author www.javadb.com
public class Main {
// Este método convierte un String a un vector de bytes

public void convertStringToByteArray() {

String stringToConvert = "This String is 15";
  byte[] theByteArray = stringToConvert.getBytes();
  System.out.println(theByteArray.length);
}

public static void main(String[] args) {
  new Main().convertStringToByteArray();
}
}
```

Listado B.2: Ejemplo de código fuente en lenguaje C

```c
// Este código se ha incluido tal cual está en el fichero LaTeX
#include <stdio.h>

int main(int argc, char* argv[]) {
  puts("¡Hola mundo!");
}
```

Listado B.3: Ejemplo de script en Matlab

```matlab
function f = fibonacci(n)
% FIBONACCI Fibonacci sequence
% f = FIBONACCI(n) generates the first n Fibonacci numbers.
% Copyright 2014 Cleve Moler
% Copyright 2014 The MathWorks, Inc.

  f = zeros(n,1);
  f(1) = 1;
  f(2) = 2;
  for k = 3:n
    f(k) = f(k-1) + f(k-2);
end
```

Algunas veces lo que se quiere ilustrar es un algoritmo o método con el que se resuelve un problema abstrayéndose del lenguaje de implementación. El paquete `algorithm2e` proporciona un entorno `algorithm` para la impresión apropiada de algoritmos, tratándolos como objetos flotantes y con mucha flexibilidad de personalización, como se observa en el algoritmo B.1 del ejemplo.

[7] Por supuesto, en un TFG (Trabajo Fin de Grado) o tesis de un centro superior de Informática.

Algoritmo B.1: Cómo escribir algoritmos

Datos : este texto
Resultado : como escribir algoritmos con LaTeX2e

```
1  inicialización;
2  while no es el fin del documento do
3      leer actual;
4      if comprendido then
5          ir a la siguiente sección;
6          la sección actual es esta;
7      else
8          ir al principio de la sección actual;
9      end
10 end
```

B.6. MENÚS, PATHS Y TECLAS CON EL PAQUETE `MENUKEYS`

Cada vez es más usual que los trabajos en ingeniería exijan el uso de software. Para poder especificar de modo elegante el uso de menús, pulsaciones de teclas y directorios, se recomienda el uso del paquete `menukeys`.[8] Este paquete nos permite especificar el acceso a un menú, por ejemplo:

Herramientas ▸ Órdenes ▸ PDFLaTeX

También un conjunto de teclas. Por ejemplo: Ctrl + ⇧ + T

O un directorio: C: ▸ user ▸ LaTeX ▸ Ejemplos

Aunque este paquete permite muchas opciones de configuración de los estilos aplicados, esto no es necesario para obtener unos resultados muy elegantes.

[8] https://osl.ugr.es/CTAN/macros/latex/contrib/menukeys/menukeys.pdf