

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

On the Automation of Machine Learning Pipelines

Alexandre Carqueja



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. João Paulo Fernandes

Co-Supervisor: Prof. Bruno Cabral

July 5, 2022

On the Automation of Machine Learning Pipelines

Alexandre Carqueja

Mestrado em Engenharia Informática e Computação

July 5, 2022

Abstract

As more and more companies start using Machine Learning models for the core technology behind their products, it becomes increasingly necessary to develop and implement efficient, and cost-effective ways to continuously produce these models.

To address the problem of continuous delivery, Machine Learning pipelines emerged. Their goal is to create efficient development environments which facilitate the deployment, evaluation, and maintenance of Machine Learning models.

The importance of employing automated Machine Learning pipelines is twofold. First, by increasing automation we are reducing the amount of time it takes to create Machine Learning models, which cuts costs in human-resource labor. Secondly, automation also reduces human errors which are bound to happen in repetitive manual tasks. Thus, the quality and efficiency of the model creation process is increased.

Our objective was to improve on existing work and create a highly extensible, lightweight, automated Machine Learning pipeline, with minimal setup overhead. In this dissertation, we have developed and tested a lightweight pipeline, capable of automating the most common and cumbersome repetitive ML tasks, while still being domain-agnostic enough to be used and expanded in any Machine Learning project.

To accomplish this goal, we have leveraged and integrated recent research and tools, such as ML Bazaar, DVC, and Pandas Profiler, to create a single end-to-end, Human-Centered, Automated Pipeline. By bridging these tools together, we minimized sources of technical debt, facilitated team collaboration, and automated repetitive tasks as much as possible.

To guide the development of our system, we have collaborated with Feedzai. Feedzai is a Portuguese Software company that employs a wide variety of Machine Learning models which classify transactions of multiple clients, as fraudulent or safe. With their insights, we were able to steer the development of our tools, in order to address the most prevalent issues usually found in the practical domain.

To evaluate the effectiveness of our proposed solution, we have compared our tools to some existing alternatives. This comparison has let us conclude that our approach significantly reduces infrastructure and setup complexity, when contrasted with the most prevalent end-to-end pipelines. Likewise, when compared to other lightweight alternatives, our tools managed provide an increase in supported features, and automated task percentage, which resulted in a more complete development process.

Keywords: Machine Learning, Machine Learning Pipeline, Pipeline Automation, MLOps, DevOps, AutoML

Acknowledgements

I would like to express my gratitude for all the people who have helped me while writing this dissertation, and for all the supported I have been fortunate to receive for these past five years of studying.

First, I must extend my thanks to my Supervisor, Prof. João Paulo Fernandes, and Co-Supervisor, Prof. Bruno Cabral. They have both been a constant source of guidance throughout the course of this dissertation, and it is due to their continuous feedback that I am able to present this work as it is, today. Likewise, I am equally grateful for the collaboration of Prof. Nuno Lourenço, and all our team members at Universidade de Coimbra, and with whom I have learned a lot from.

Of course, none of this would have been possible without the knowledge and opportunities I've been given by my University, as such, I would like to congratulate everyone at FEUP. Specially my professors who have taught and amazed me for these past five years.

Lastly, and most importantly, I must give a big thank you to my family and friends, who have persistently supported me throughout the course of this Masters degree, specially during the hardest of times.

Sincerely, Alexandre Carqueja

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Dissertation Context	3
1.3	Objectives	3
1.4	Document Structure	4
2	Background and Related Work	5
2.1	Background in MLOps	5
2.1.1	An introduction to DevOps	5
2.1.2	The Birth of MLOps	6
2.1.3	Data Versioning	8
2.1.4	Data Validation	8
2.1.5	Model Versioning and Experiment Tracking	9
2.1.6	Model Deployment and Monitoring	9
2.1.7	Understanding Technical Debt	10
2.2	The state of MLOps	11
2.2.1	Manual Workflow Shortcomings	11
2.2.2	MLOps Tools	12
2.2.3	MLOps Pipelines	14
2.2.4	MLOps level of adoption	15
2.3	Background in AutoML	16
2.3.1	Hyper-parameter Optimisation	17
2.3.2	Pipeline Search	17
2.4	The state of AutoML	18
2.4.1	Novel Approaches	18
2.4.2	Performance of AutoML Systems	19
2.4.3	The usefulness of AutoML	20
2.5	State of the Art Conclusions	21
2.5.1	Human-Centered Automation	22
2.5.2	How Machine Learning Pipelines can help with Debt Management	23
2.5.3	Data-Centric vs. Model-Centric	24
2.5.4	The need for Lightweight Solutions	25
3	Our Approach	27
3.1	Chosen Representation for Machine Learning Pipelines	27
3.2	Quick Machine Learning Framework Architecture	29
3.3	QML Framework capabilities	30
3.3.1	QML Pipeline Initialization	32

3.3.2	QML Watchdog	33
3.3.3	QML Commands	34
3.3.4	Creating vs. Initiating an Environment	35
3.4	Lightweight Pipeline Architecture	35
3.4.1	Data Management Phase	36
3.4.2	ML Preparation Phase	37
3.4.3	Model Building Phase	38
3.4.4	Deployment Phase	38
3.5	Lightweight Pipeline Demonstration	39
3.5.1	Data Management Phase	39
3.5.2	ML Preparation Phase	41
3.5.3	Model Building Phase	41
3.5.4	Deployment Phase	42
3.5.5	Implementation Conclusion	42
4	Experimental Validation	45
4.1	Comparison Introduction	45
4.2	DAGsHub Analysis	46
4.2.1	Set-up Process	46
4.2.2	DAGsHub Experiment	47
4.2.3	QML vs. DAGsHub	49
4.3	Kubeflow Analysis	52
5	Discussion	55
5.1	Conclusions	55
5.2	Future Work	56
5.2.1	Future Work on the QML Framework	56
5.2.2	Future Work on the Lightweight Pipeline	57
5.3	Closing Note	58
A	Lightweight Pipeline Screenshots	59
A.1	Inspecting Data	59
A.1.1	Automatically Generated Report	59
A.1.2	Deployed Model Test	59
	References	63

List of Figures

2.1	DevOps lifecycle, adapted from [37]	7
2.2	MLOps lifecycle, adapted from [71]	8
2.3	Manual ML pipeline as presented in [30]	11
2.4	(left) The impact of increasing number of total GPUs assigned to workers on time to process one epoch of training. (right) Cost of running one epoch on the Google Cloud Platform, with Reserved vs. Preemptible resources. As explored in [22]	13
2.5	MLOps Framework proposed in [24]	14
2.6	MLOps Maturity Model proposed in [24]	15
2.7	Pipeline types defined in [39]	18
2.8	Composite pipeline results as shown in [39]	19
2.9	Comparison of different Hyper-parameter optimization techniques performed in [33]	20
2.10	Proposed conceptual framework: a reference interactive VA/ML pipeline is shown on the left (A–D), complemented by several interaction options (light blue boxes) and exemplary automated methods to support interaction (dark blue boxes). Interactions derive changes to be observed, interpreted, validated, and refined by the analyst (E). Visual interfaces (D) are the “lens” between ML models and the analyst. Dashed arrows indicate where direct interactions with visualizations must be translated to ML pipeline adaptations. As exposed in [6]	23
3.1	General Machine Learning Workflow derived from [32], [30], and [28]	28
3.2	Architecture of our proposed solution. At the top, the developer interacts with QML, which then utilises the selected pipeline, in this case, the Lightweight Pipeline.	29
3.3	Diagram of our proposed Lightweight Pipeline.	36
3.4	Enlarged diagram of the Data Management phase, in the Lightweight Pipeline.	36
3.5	Enlarged diagram of the ML Preparation phase, in the Lightweight Pipeline.	37
3.6	Enlarged diagram of the Model Building phase, in the Lightweight Pipeline.	38
3.7	Enlarged diagram of the Deployment phase, in the Lightweight Pipeline.	39
3.8	Diagram for the steps in our Machine Learning model.	43
4.1	Storage Architecture of DAGsHub, presented in [20]	47
4.2	Pre-built pipeline generated by DAGsHub "Cookie Cutter DVC Template", in our repository.	48
4.3	Data exploration tool provided by DAGsHub, in our repository.	48
4.4	Plot of class distribution in wine quality data-set.	49
4.5	Plot of correlation Matrix in wine quality data-set.	49
4.6	Experiment Tracking of our Wine Quality Project in DAGsHub Web UI.	49
4.7	Workflow Diagram proposed for DAGsHub project development.	50
4.8	Kubeflows component Architecture, as presented in [43]	53

A.1	Example of a data report automatically generated when a data file is added or modified.	60
A.2	Example of a full report generated with our inspect_data command.	61
A.3	Example of a live prediction made after the deployment of our Wine quality model, built with the Lightweight Pipeline.	62

List of Tables

2.1	Comparison between Model-Centric and Data-Centric approaches derived from [19]	24
3.1	Table with Model Scores.	42
4.1	Setup and infrastructure comparison between QML and DAGsHub	50
4.2	Feature and automation comparison between QML and DAGsHub. "Not Supported" for tasks that are not at all supported by the pipeline. "Assisted" for tasks that are mostly manual, but have some pipeline support. "Semi-Automated" for tasks that are mostly automated but require some sort of user input or direction. "Automated" for tasks that are completely automated and require no user direction.	51
4.3	ML Model results comparison between the Lightweight Pipeline and DAGsHub's pipeline.	52

Listings

3.1	Example of an installed QML pipeline directory structure.	30
3.2	Example of a specification file used by QML to describe a pipeline.	31
3.3	QML's "start" command options.	32
3.4	Example of a setup process.	33
3.5	Example of an event handler file.	34
3.6	Example of a command file from our pipeline.	34

Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
SLA	Service-Level Agreement
CLI	Command-Line Interface
QML	Quick Machine Learning
YAML	yet another markup language
DAG	Directed Acyclic Graph
DAG	Directed Acyclic Graph
AutoML	Automated Machine Learning
CASH	Combined Algorithm and Hyper-parameter Selection
EDA	Exploratory Data Analysis
API	Application Programming Interface

Chapter 1

Introduction

Machine Learning (ML) is the sub-field of Artificial Intelligence (AI), which has been in the spotlight for the past few years. Between 2019 to 2020 this area of research saw a growth of 34.5% in journal publications, almost doubling that of the year prior [51]. With these advancements, many challenging problems are being tackled, such as personalised recommendation systems [10], and autonomous driving [7].

Due to this rise in popularity and usage, more and more companies have started relying on AI as a core element for their business models, and in turn, on Machine Learning. Such evidence can be found in McKinseys 2019 report, which saw a 25% year-over-year increase in AI adoption by companies, in their processes [49].

The wide-spread adoption of AI technology has introduced the need for efficient methods to continuously deliver, improve, and deploy ML models. A need that is exacerbated by the current difficulties companies face with Machine Learning adoption, which is often deemed very costly and time consuming, as reported by a 2019 survey on Machine Learning deployment time [11].

The aforementioned difficulties have contributed to the creation of MLOps, a new field of research centered around Machine Learning Operations. MLOps is an adaptation of the decade-old field of DevOps, which was created to address the analogous concerns of continuous deployment, but for "standard" software [27]. Unlike DevOps, however, MLOps seeks to address Machine Learning specific issues, such as model and data versioning, continuous training, model monitoring, and model testing.

Despite the successes that MLOps has had, the need to make Machine Learning more accessible spurred the growth of AutoML. The main goal behind AutoML research is to automate the entirety of the Machine Learning process, making it, therefore, accessible to anyone without the need for extensive knowledge in Data Science [15]. And while AutoML research is still a hot topic, far from reaching its ultimate conclusion, there have already been some interesting advancements which we will leverage throughout this dissertation.

1.1 Motivation

Developing and maintaining Machine Learning systems is difficult and costly, which often leads to failed or abandoned projects, as can be seen in a report by the IDC in 2019 [13]. In this report, a survey was conducted in which the respondents, companies developing enterprise-wide AI solutions, reported up to 50% failure rate for AI projects. Among the reasons cited, the most common were high costs, and a lack of qualified professionals.

On a more recent 2021 report [62], 508 Machine Learning practitioners were surveyed, and 68% of the respondents admitted they had abandoned 40 - 80% of all ML experiments conducted in the previous year.

While it may be argued that this is a self-solving problem, as every year more and more qualified professionals enter the workforce, that argument does not paint a full picture. It is also worth noting that, as ML is further adopted by a larger number of institutions, the need for Machine Learning specialists also rises, and that gap between demand and supply does not seem to be decreasing [57].

As an alternative to increasing the number of trained professionals, it is possible to reduce the cost of Machine Learning development by simplifying the development process. Furthermore, this approach also has the added benefit of decreasing the management overhead in ML projects. The reasons being that, a simpler workflow is easier to organize, and also because, if smaller teams are able to accomplish more ambitious projects, the amount of people that need to be managed is reduced.

Another incentive for researching ways to simplify and automate ML pipelines is the push for the democratization of AI. While, theoretically, anyone with the required knowledge could start working on a Machine Learning project, the resources required to create, deploy, and maintain an effective AI solution are much more costly than just a personal computer. As such, it is natural that most of the developments presented in this field are contained in the realm of large corporations, or heavily funded institutions.

Currently, the existing ML pipelines employed by the industry make use of complex and expensive infrastructure, such as high performance cloud environments. This approach provides many benefits for cutting-edge ML research, however, it also comes with large maintenance costs, and unwanted setup overheads. Aside from that, in most cases, these pipelines provide little to no extensibility support, meaning that if a use-case is not covered by the chosen pipeline, it is often necessary to increase the infrastructure's complexity by adding a separate component to it.

By simplifying this process, we are, once again, making it easier for smaller teams with limited access to resources to pursue ambitious ML problems. Lastly, by reducing the complexity of Machine Learning development, we hope not just to broaden the adoption of the technology, but also to encourage higher quality solutions, as simpler workflows lead to less mistakes, and thus, better results.

1.2 Dissertation Context

In this dissertation, we will be applying and expanding on both MLOps and AutoML concepts, with the goal of developing a highly extensible, automated, Machine Learning Pipeline.

On one side, the wider context in which this dissertation finds itself, is that of an industrial landscape where, despite the exponential adoption of Machine Learning systems, Machine Learning Pipelines have yet to see the same kind of enthusiasm [26]. As such, it is a central aspect of this dissertation to advance Machine Learning Pipelines in the direction that best fits the needs of the wider audience, encouraging further adoption by making implementation simpler and more accessible.

On the other side, this dissertation has been developed as part of the CAMELOT project, a research initiative co-promoted by Universidade de Coimbra, Instituto Superior Técnico, Faculdade de Ciências da Universidade de Lisboa, Carnegie Mellon University, and Feedzai, with the goal of furthering the research on automated pipelines and anonymized data in Machine Learning. As such, it is important to note that this work is partially funded by the European Social Fund, through the Regional Operational Program Centro 2020 and by the CMUIPortugal project CAMELOT (POCI-01-0247-FEDER-045915).

One of the main collaborators in this dissertation, Feedzai, is a Portuguese software company that uses Machine Learning models to classify monetary transactions as fraudulent or valid. As such, the company works with many different partners (banks, physical retailers, online retailers, etc.), and, for legal reasons, it must not use data from one partner to train the models of another. Additionally, Feedzai is always aiming to improve the accuracy and functionality of their products, which results in very frequent online updates of their services.

With all of these factors combined, we end up with an environment which has a very large number of ML models that must be continuously trained and deployed by the company. This makes it crucial for Feedzai to have a strong understanding of their underlying pipeline, which is why this setting proved to be the perfect environment for developing an extensible and robust ML pipeline, capable of handling real-world problems.

1.3 Objectives

The goal of the research conducted in this dissertation is to simplify the Machine Learning workflow, by taking advantage of novel advancements in the fields of AutoML and MLOps. Additionally, it is our intention to broaden the adoption of Machine Learning pipelines, by creating a solution that can be adopted from the very beginning of any project, without a significant overhead, and without compromising future extensibility.

More concretely, the purpose of this dissertation is to answer two research questions.

RQ1: What are the required functionalities of a Machine Learning pipeline?

To answer this question, we conducted a literature review on MLOps and AutoML fields, from which we drew our conclusions, in chapter 2.

RQ2: How feasible is it to create modular pipelines that automate time-consuming Machine Learning tasks, without significantly increasing infrastructure complexity?

To answer this question, we began by creating a flowchart representation of the entire Machine Learning workflow, which helped us define the sequence of the pipeline steps. Subsequently, we developed a modular Machine Learning pipeline that addresses the identified tasks, using open source software and previous research, such as ML Bazaar, DVC, and Pandas Profiler, while remaining light on infrastructure. Lastly, we validated the suitability of our approach by applying the developed pipeline to a standard ML problem.

RQ3: How does a lightweight pipeline perform in comparison to existing alternatives?

To answer this question we compared our pipeline to existing solutions, by re-running the same experiment on several tools. With this approach, we concluded upon the advantages in setup-overhead and infrastructure reduction, as well as the disadvantages of our approach.

1.4 Document Structure

In chapter 2, we provide a review for the State of the Art in the area of Machine Learning pipelines. This Chapter is subdivided in 2 topics, one for MLOPs, and another for AutoML, in which we expose some necessary background technology, and the latest developments in both of these fields.

In chapter 3, we present the bulk of our work in this dissertation. First, we go over our proposed representation for Machine Learning Pipelines and Workflows. Then, we explain how we have subdivided our development into two tools, and proceed to explore the inner workings of the first one (QML). Afterwards, we explain how we have leveraged QML in order to create our Lightweight Pipeline, and provide a demonstration for it by solving a simple Machine Learning project.

In chapter 4, we validate our proposals, by comparing against two other existing solutions in the field of Machine Learning Pipelines. And lastly, in chapter 5 we conclude upon the adequacy of our research in the Machine Learning development space, and on the limitations that it currently faces.

Chapter 2

Background and Related Work

In this chapter, we will expose the literature review we performed in the relevant fields of MLOps and AutoML, as well as the necessary background information. This research served to inform the decisions made throughout the development process, as well as to direct the objectives of our work towards the most relevant areas.

Below, we can find a section dedicated to exploring the foundations of MLOps (2.1), followed by a review of recent research (2.2), which will explore the entire structure of the pipeline-driven Machine Learning Process, as well as advancements that have been made in the management of each stage of the process.

In Section 2.3, background information on the AutoML field was exposed, followed by a discussion on recent developments on the field of AutoML in Section 2.4, starting with the overall effectiveness of AutoML, an analysis of the different available tools for AutoML, and the needs that practitioners face regarding AutoML systems.

Lastly, we conclude on the key findings of this literature review and set the stage for the next chapter in Section 2.5.

2.1 Background in MLOps

In order to introduce the current developments in the field of MLOps, we will first go over some of the basic background concepts in this area, so that the following articles can be analysed with the proper context in mind.

To that end, we will first introduce the reader to MLOps "predecessor", DevOps, and then we will explain some of the key differences between the two, as well as the major concerns that need to be taken into account.

2.1.1 An introduction to DevOps

The term DevOps was first introduced around 2009 in a series of talks about joining the Development and Operations teams, to increase productivity and product quality [8].

At the time, the standard practice in the industry was to keep software development separate from the deployment and maintenance operations. However, this separation often led to tension and conflict between the two groups, as each one had different objectives and priorities, which regularly resulted in a series of problems, such as:

- Slow software delivery cycles.
- Bug-ridden products.
- Unpredictable deployments, with many last minute problems.

The goal of DevOps was primarily to bring these two groups together, and have them work towards a common goal. This, of course, required a drastic culture shift throughout the industry, as well as new management practices and procedures, but as the change occurred, several benefits started to become evident.

This novel development environment soon gave rise to a collection of many different tools, such as automation, monitoring, and configuration management tools. All of which helped standardise, automate, and facilitate the development process, thereby greatly shortening software delivery cycles, as well as ensuring seamless deployments, and enforcing best practices, such as automated testing.

In the end, all of these improvements can be distilled into two central aspects:

- Continuous integration (CI);
- Continuous deployment (CD);

CI is the software development practice which ensures that team members integrate their work frequently, so as to keep everyone as updated as possible. These integrations must always be subject to quality assessments, such as automated testing, and automated building.

CD is the software engineering approach in which new full updates to the software are automatically deployed without the need for any human intervention. This, coupled with the short development cycles, leads to a constant stream of new deployments, sometimes as frequent as several times per day [3].

Over the last decade, DevOps has been adopted as a standard practice within the software development community, and is now best known for its iconic life-cycle diagram depicted in Figure 2.1, which, if followed correctly, ensures the upholding of CI and CD, as well as the constant improvement of its processes [4].

2.1.2 The Birth of MLOps

Following the rise in Machine Learning popularity, it was soon evident that existing DevOps practices were not enough to accommodate for ML's particular needs. As such, MLOps emerged as an adaptation of DevOps concepts, adjusted and extended in order to fit the needs of complex Machine Learning environments.

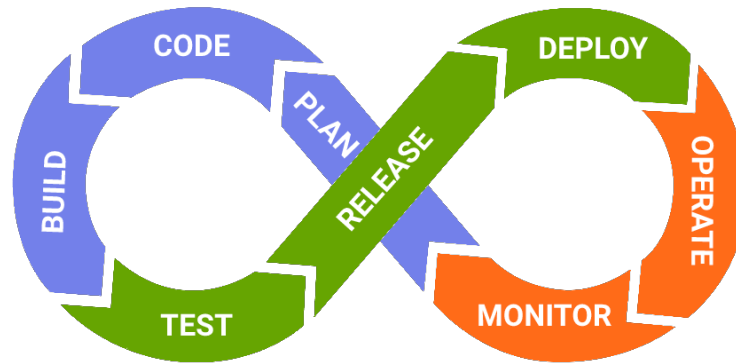


Figure 2.1: DevOps lifecycle, adapted from [37]

MLOps aims to answer all of DevOps challenges, keeping the practices of CI and CD, but while also introducing some new concerns: [29], [18]

- Data Versioning
- Data Validation
- Model Versioning
- Experiment Tracking
- Model Deployment
- Model Monitoring

With these additional changes, the previous DevOps workflow no longer fit to this new requirements, and so new diagrams were created. As of the time of this writing, there does not seem to be a consensus among the Machine Learning community towards what is the most appropriate depiction of this new life-cycle, however, we have identified a suitable representation for this dissertation, which can be seen in Figure 2.2.

To provide further context, in the following subsections each one of the new MLOps stages will be explained and discussed.

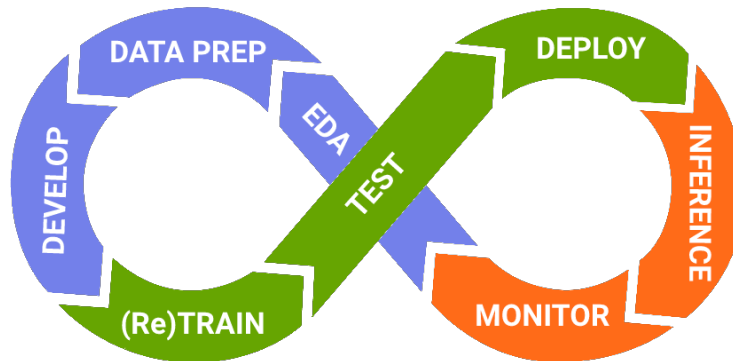


Figure 2.2: MLOps lifecycle, adapted from [71]

2.1.3 Data Versioning

Just as Code Versioning keeps track of code changes in traditional software development projects, in Machine Learning projects, Data Versioning aims to keep track of changes to the data-sets used throughout development.

It is important to keep a record of these changes, as the quality of our solution is directly dependent upon the quality of our data. By keeping track of its different versions, we are able to single out changes in performance, as well as go back to a previous version, if desired.

Unfortunately, normal Code Versioning approaches are insufficient when dealing with data files, which is why a separate tool must be employed. This has to do with the traditionally larger file sizes, as well as differing structures between code files and data files. Furthermore, by keeping these two aspects separate, it becomes possible to test an older version of code with new data, and vice-versa.

2.1.4 Data Validation

Once again, it is possible to draw a parallel between the validations performed on code, in traditional software development, and the validations required by data files, in Machine Learning problems.

In the same way that, when there is a change in a code-base, we must assure that previously working features are still functional. When there is a change in a projects data-base, the new data file must be tested, in order to ascertain that it complies with the expected properties of its predecessors.

In some cases, this might be as simple as making sure that all files follow the same structure, with the same amount and names of features. However, due to the diverse array of domains in which Machine Learning is applied, these tests can take on many forms. Such as guaranteeing certain statistical properties in some of the data-sets features, making sure that the target classes are properly represented, checking whether annotations follow the required format, etc.

All of these automated checks are useful, as they systematically test each property without fail, thus reducing the amount of manual work required per new data-set, and immediately alerting the developer towards what might have gone wrong on a given data update.

2.1.5 Model Versioning and Experiment Tracking

Akin to Data Versioning, Model Versioning takes care of storing and versioning all Machine Learning models. Just as with previous versioning processes, this brings the same benefits of quickly changing between versions on a given model, which might be useful when rolling back a deployment due to an unexpected error. However, it can also be used to compare the performances on different versions of the same model, which is most useful when paired with an Experiment Tracking tool.

Experiment Tracking tools are mostly used to save all of the relevant information that a developer might need when inspecting a given trained model. Such as, the data-set used during training, the selected hyper-parameters, the validation score, the test score, the resources utilised during training, etc.

The goal of displaying all of this information is to help ML professionals to understand what approaches have already been tried, and what should be attempted next, in order to improve the models performance.

2.1.6 Model Deployment and Monitoring

Lastly, when a sufficiently good model is trained, it is necessary to deploy in some way or another. The most common approach is to deploy the model as a service, with a Rest API that returns a prediction when given an input. But, depending on the problems context, other different deployment strategies might be desired, such as embedding the model on an existing application. Regardless of the deployment strategy, however, it is always necessary to uphold DevOps CD principle, by making deployment automatic and continuous.

After successfully deploying a solution, it is necessary to make sure that it is performing as expected. This means that besides producing a prediction, our system must also save the given inputs and, if possible, determine whether each prediction was correct or wrong. With this newly collected data, it is then possible to monitor the system and alert the developers, in case any evidence of performance deterioration, or drift, is found.

2.1.7 Understanding Technical Debt

As we will see in the following sections, one of the big challenges in developing large and long-lasting software projects, including Machine Learning systems, is dealing with technical debt. But first, what is technical debt?

As the name implies, and much like monetary debt, technical debt represents a cost that is introduced at a given point in the development of software engineering products, but which will only be paid for later down the line, and with added interest.

To translate the metaphor into a more practical setting, it is useful to understand that technical debt can take on many forms. Such as: Architecture Debt, Code Debt, Documentation Debt, Infrastructure Debt, etc. [68] Some concrete examples of these debt sources are:

- Duplicated code
- Dead code
- Glue code
- Outdated or nonexistent documentation
- Lack of testing
- Lack of automation
- Non-uniform naming conventions
- Tightly coupled architecture

Even from this small sample of examples, it becomes apparent that there is a lot of variety in technical debt sources, and while the specific cause and solution for these problems may differ, the main reason for their existence is the same.

Usually, whether consciously or unconsciously, these issues arise from a choice to prioritize faster progress over perfect execution. It might be due to lack of expertise, lack of resources, or even a deliberated choice to release an imperfect solution now, rather than taking the necessary time to develop a more robust code-base.

While the concept of technical debt is very much applicable to Machine Learning systems, traditional approaches to solve this problem are not sufficient when working in an ML setting. This particularity is due to the new debt sources that come from data and models in ML projects, as well as the new technical debt categories that arise from them. This problem, and recent advancements in how to best deal with it, will be further discussed in the next section, as well throughout this dissertation, since technical debt plays an important role in pipeline design.

2.2 The state of MLOps

After understanding the context of MLOps, which was explored in the previous section, we will now go over some of the most recent advancements in this field. As such, in this section we will provide a complete picture of what is being researched, what has been accomplished, and what is still missing.

2.2.1 Manual Workflow Shortcomings

In most experimental projects, the Machine Learning workflow is not pipeline-driven. In these cases, the goal is simply to develop a single model as a proof of concept. As such, any regards for continuous training, automated deployment, model versioning and monitoring, can be disregarded. In this basic use-case we are left with a workflow such as the one in Figure 2.3. This simple approach, however, leaves many operational problems unaddressed.

In 2015, Googles highly influential paper on the Hidden Technical Debt of Machine Learning Systems [2], Sculley et al. exposed some of the major factors behind maintainability hardships in Machine Learning Systems. In this paper, many reasons for technical debt were presented, such as Entanglement, Correction Cascades and Data Dependencies, which affect Machine Learning Systems under the CACE principle (Changing Anything Changes Everything). Additionally, other forms of technical debt were explored, such as Glue Code, Pipeline Jungles and Dead Experimental Codepaths, which are more related to integration issues. Lastly, other problems, such as common Code Smells, Configuration Debt, and other ML-related Debt causes, were exposed, and possible solutions were discussed.

Since then, there have been some efforts to reduce the impact of these issues, as can be seen in a more recent study by Tang et al. [31], which conducted a survey on 26 ML Projects and analysed the refactoring done within those projects. Among the findings of this paper it was shown that only 3 of the issues exposed by Sculley et al. were being consistently addressed (configuration debt, plain-old-data type, and multiple language debt). Furthermore, a taxonomy with 14 elements for new ML-specific refactoring techniques was proposed, as well as 7 new technical debt categories. Finally, some further recommendations were given in order to combat technical debt in Machine Learning Systems.

As can be seen from both of these papers, Machine Learning maintainability is no trivial matter, and well designed pipelines can help tackle these issues.



Figure 2.3: Manual ML pipeline as presented in [30]

2.2.2 MLOps Tools

In the MLOps field, there are essentially two types of available tools. Full end-to-end Machine Learning pipelines, that take care of the entire project life-cycle in a single coherent solution. And, isolated tools developed for specific tasks within the life-cycle, which do not take into account the remaining infrastructure, but that must then be integrated into whatever custom pipeline the developers are using.

The argument in favor of full end-to-end solutions is that the teams need not waste resources building their own pipeline, and can get straight into the Machine Learning process. The price paid for this advantage is monetary, since most solutions are provided as paid services, but also flexibility, as the developers become locked to whichever tools the pipeline provides, with no way of adapting it to their specific needs.

There are many options when choosing between full Machine Learning pipelines, such as AWS SageMaker [59], Pachyderm [34], Valohai [69], CNVRG [48], and many more [41].

One option that differentiates itself from the rest, however, is Kubeflow [54], both because of its open-source nature, which is rare amidst the other alternatives, as well as for the ability to create add-ons, which makes it more extensible than the rest.

One successful example of a Kubeflow implementation can be found at CERN, as described by Dejan Golubovic and Ricardo Rocha in their 2021 publication [22]. In this article, the authors describe CERN's need for a scalable solution, capable of serving the many different problems being tackled there, without the need of an overly-large body of engineers to maintain its infrastructure.

Given Kubeflow's characteristics it was chosen as the foundation for CERN's ML pipeline. As it runs on top of Kubernetes, CERN deployed it on a large private cloud that was already operational, which provides scalability benefits, as well as the ability to adapt resource consumption to what is required by each process at any given time.

Development with Kubeflow is done remotely on CERN's self-hosted cloud, by making use of Jupyterlab instances, which allow developers to utilise Jupyter Notebooks for their experiments. Kubeflow's KALE extension then facilitates the translation of the sequential Jupyter notebooks into containerized pipelines that are run on several nodes in the network. Additionally, CERN also takes advantage of automated hyper-parameter optimization and distributed training through Kubeflow's base features and Katib extension.

Lastly, Kubeflow deploys the resulting models through an exposed, serverless, REST API, built on Knative, which is able to auto-scale resources as required by the current number of requests [52].

For the validation of this setup, a sample 3DGAN project was utilised, with which it was made apparent that solely relying on CERN's private cloud did not provide the necessary resources for fast training. As such, Google's public cloud was also utilised, and a cost analysis was performed. Here the authors concluded that the increase in performance was nearly linear to the increase in resources, and that, through use of Preemptibles (Google Clouds low SLA instances), it was possible to significantly cut on Google Cloud resource costs, without a significant hit to performance,



Figure 2.4: (left) The impact of increasing number of total GPUs assigned to workers on time to process one epoch of training. (right) Cost of running one epoch on the Google Cloud Platform, with Reserved vs. Preemptible resources. As explored in [22]

as shown in Figure 2.4.

In the end, it is easy to see why end-to-end pipelines are such a powerful option, as they come ready with a myriad of useful features. However, it is also the case that not all Machine Learning projects require such intricate environments, and that the complexity and infrastructure cost might be too high for smaller teams. In this case, hosting and maintaining a private cloud system with Kubernetes, or utilizing a public service such as Google Cloud can quickly get outside the budget of many startups and small research groups.

Aside from the multitude of end-to-end pipelines available, there have also been some advancements in standalone tools that facilitate Machine Learning development.

In [17], Smith et al. proposed a novel framework for ML pipeline building which addresses some of the technical debt issues discussed in previous papers.

This proposed system aimed to standardize the integration of many different ML libraries and custom-built functions with a common API, in a framework entitled ML Bazaar. At the foundation of this system lies a primitive-based approach, where many existing ML functions from various libraries are annotated in a JSON file, with the necessary metadata to be successfully joined together.

By doing this, it became possible to integrate modules from different libraries, as well as contribute new modules, without the need for any glue code. Furthermore, by simply providing ML Bazaar with an ordered list of which modules to use, it is possible to create modular pipelines that can be easily iterated upon and expanded.

Additionally, this ecosystem was extended with AutoML capabilities, which rely on search algorithms based on Gaussian Process, to tune hyper-parameters and find optimal pipeline structures.

Lastly, their system was successfully implemented in 5 different case-studies, and their AutoML system was evaluated in an AutoML competition where it placed second from the 10 competing teams.

One limitation of tools like this, is that they only tackle a select few steps of the Machine

Learning workflow, and must be integrated with others to create a comprehensive tech suite capable of addressing all the challenges a team might face.

2.2.3 MLOps Pipelines

In [24], John Meenu Mary et al. conducted a literature review on the field of MLOps, which they used to support the proposal of a framework for MLOps infrastructures, as well as a maturity model for MLOps adoption in the deployment phase.

Based on their findings from the conducted research, John et al. proposed a framework composed of 3 Pipelines. One pipeline for handling data, another for developing and testing Machine Learning models, and a final one for deploying and monitoring the solution, as depicted in Figure 2.5.

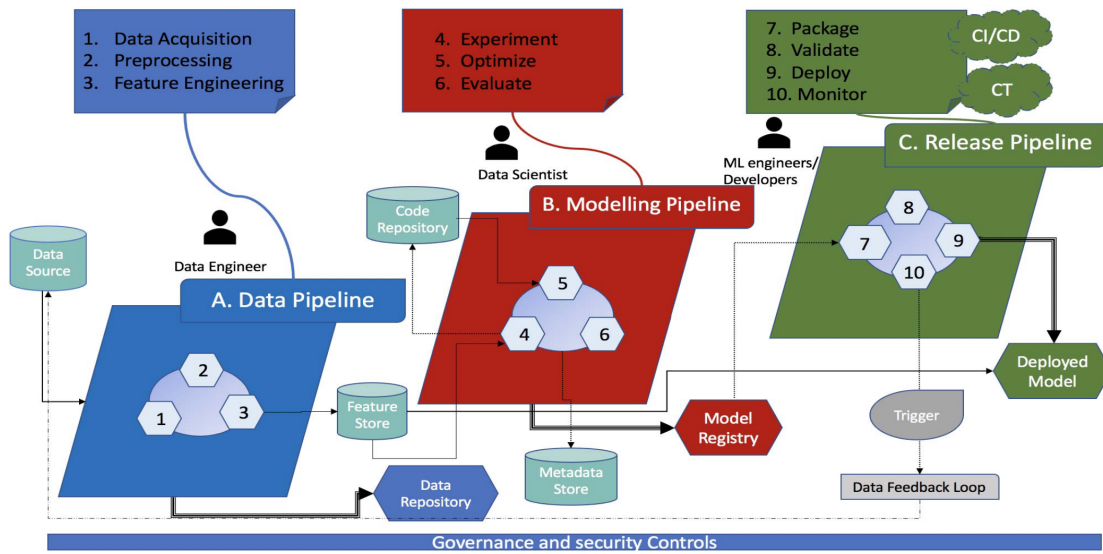


Figure 2.5: MLOps Framework proposed in [24]

This framework, as well as the maturity model depicted in Figure 2.6, proved to be adequate in the 3 companies in which they were validated, fitting the companies practices as expected. However, it did not account for less matured environments, where MLOps practices are not yet fully integrated, and where teams might have overlapping responsibilities across the pipelines.

In the end, this study was able to successfully identify the benefits of MLOps and highlight the need for such systems. However, the simplistic nature of their proposed Maturity level assessment, makes for an incomplete categorization scheme, as it leaves out many other aspects of MLOps adoption, such as automated EDA, automated versioning, automated model re-training, etc.

In another article [30], Ruf et al. conducted an overview of DevOps and MLOps principles, following with a proposed representation for a typical MLOps workflow, comprised of 4 different phases, each with several stages (Data Management, ML Preparation, ML Pipeline, and Deployment Phases), as well as 6 different actors (Data Scientist, Domain Expert, Data Steward, Data

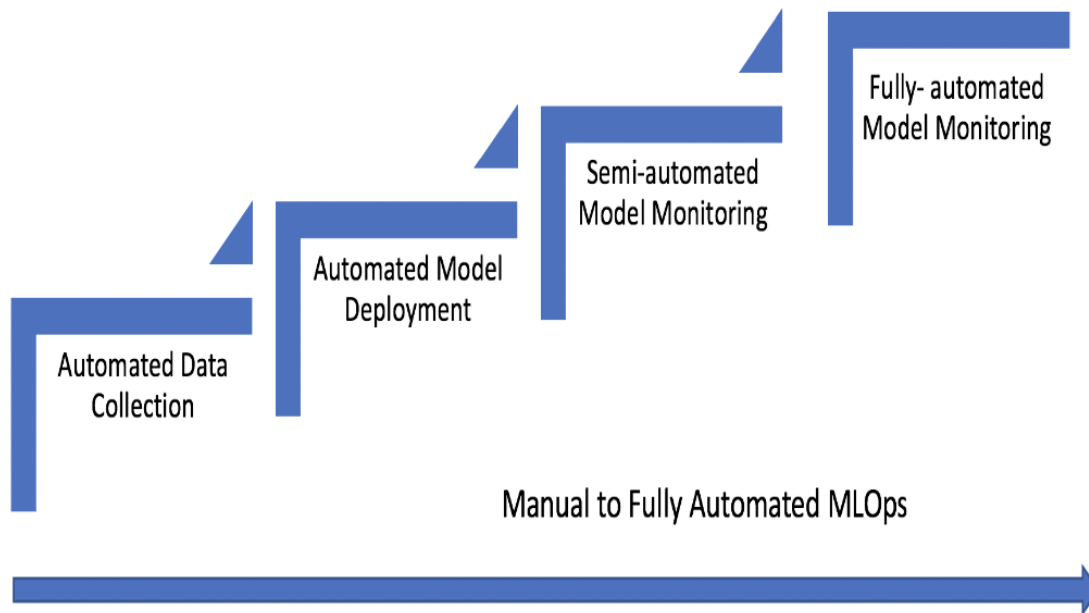


Figure 2.6: MLOps Maturity Model proposed in [24]

Engineer, SW Engineer, MLOps Engineer). More crucially, 26 open-source tools were compared and benchmarked in accordance to how many of the different stages they support and how well they do it. Finally, a small example is given, of an MLOps pipeline using MLFlow, Git and DVC for object detection with images.

Lastly, in yet another study [28], Paleyes et al. performed a review on published reports of Machine Learning deployment solutions. From the analysed case-studies, a list of challenges practitioners face at each stage of the pipeline was created, and for each challenge some possible solutions were discussed.

As a closing note, it is important to note that the pipeline schemes presented by the analysed papers all differed in their structure, even if some aspects did overlap. Due to this dissimilarity, we designed our own comprehensive workflow diagram, which will be presented in chapter 3, and used it to guide the development of our Machine Learning Pipeline.

2.2.4 MLOps level of adoption

It is difficult to gauge, exactly, how much of the industry has already adopted MLOps practices, however, in [26], Mäkinen et al. conducted a survey aiming to answer this exact question. In this survey, 331 professionals from 63 different countries reported on their Machine Learning experience, projects, and needs.

One of the major takeaways from this survey, was that most of the respondents were working on the early stages of Machine Learning. Such as, collecting data, figuring out how to use the data,

learning Machine Learning technologies, and developing new models. Taking this surveys sample as representative of the industry, we can conclude that most professionals still find themselves working in experimental Machine Learning projects, where pipeline-driven development is not yet a concern. In fact, only 22 out of the 331 respondents indicated pipeline infrastructure as a main concern.

Additionally, up to 40% of respondents said they worked both on model development as well as infrastructure management, which might be due to the large percentage of small experimental projects, and lack of additional resources. This, for the first time, introduces the still ambiguous concept of a "Full-Stack" Machine Learning developer.

Finally, another insightful discovery, was that, as companies move away from proof-of-concept use of Machine Learning, and start integrating the technology more into their core business, concerns with frequent deployments and pipeline development rise.

Despite that, the key challenges identified by the respondents were still centered around data related issues, and project scope. Such as, lack of data, messiness of data, and also, unclear or unrealistic expectations.

Interestingly, these types of problems are not issues that arise during development, instead, their causes are already present since the inception of the project. This denotes an evident lack of clarity in the respondents Machine Learning projects, perhaps due to the novelty of the field. Taking this into account, we hope that, by broadening the adoption of Machine Learning Pipelines, we can provide direction in these projects, and ease the aforementioned pain points.

2.3 Background in AutoML

As we have seen before, integrating Machine Learning into the core business strategy of a company can be a very complex problem. This complexity associated with ML development gave rise to the field of AutoML, which was first introduced around the 1990s, posing the be the "quiet revolution" of AI [14].

The main goal of AutoML is to automate certain tasks of the Machine Learning workflow, and ultimately, automate the full Machine Learning life-cycle. From data acquisition, to model deployment, monitoring, and improvement.

There are three main reasons for developing and employing AutoML. First, by utilising AutoML, it is possible to save on resources, such as time and labor, given that less manual tasks are needed.

Secondly, with AutoML we can systematically search for the best solution in a wide array of possibilities. This is most prominent in Hyper-parameter optimisation problems, where automated methods are usually more accurate than human intuition.

Lastly, as AutoML research advances, thereby making Machine Learning development simpler and more automated, the barrier to entry also reduces, such that it becomes increasingly possible for less experienced developers to develop successful Machine Learning systems, thus contributing to the democratization of AI.

2.3.1 Hyper-parameter Optimisation

In Machine Learning models, it is often necessary to decide upon a set of hyper-parameters before training the data on a given algorithm. These hyper-parameters can greatly influence the behaviour of the chosen model, altering properties like learning rate, number of epochs, regularization, model initialization, etc [5].

Traditionally, the process of choosing the most suitable hyper-parameters is done with a lot of trial and error, repeating training sessions repeatedly until we find an acceptable validation score.

AutoML has attempted to solve this problem automatically, by continuously searching through the possible hyper-parameters space and saving the options that best optimize the validation score. There are two basic approaches for doing this, the first is to use a grid search, which involves defining a list of options for each hyper-parameter and exhaustively trying out all possible combinations until we reach the best one. Another option is to use random search, which involves randomly sampling from the space of possible hyper-parameters, according to a distribution function, and choosing the best one [23].

Both of these methods can be very time-consuming, as a large number of training sessions will be necessary, so some other options were developed. One such approach is the Bayesian optimisation algorithm, which involves using previous attempts to construct a probabilistic model that maps hyper-parameters to possible objective function scores, and uses this to guide the search for the best set of hyper-parameters.

To implement Bayesian optimization, it is necessary to choose a function that maps the hyper-parameters to a score, one popular approach is to use Gaussian process. This involves fitting a Gaussian process to the data and using it to predict the value of the objective function at any point in the space of possible hyper-parameters. The algorithm then chooses the next set of hyper-parameters to try based on where it thinks the optimum is likely to be.

Another popular algorithm is the tree of Parzen estimators. This works by constructing a tree of probability density functions, each of which approximates the objective function in a small region of the space of possible hyper-parameters. The algorithm then chooses the next set of hyper-parameters to try based on which region of the space is currently being explored [9].

2.3.2 Pipeline Search

When working on a Machine Learning problem, it is not just necessary to decide upon the hyper-parameters to use when training, it is also necessary to select the algorithm we want to train our model with. In the field of AutoML, the problem of automatically choosing the best algorithm is usually entitled Algorithm Selection. This problem is usually solved in tandem with Hyper-Parameter optimization, and the joining of this two is normally referred to as a CASH problem (Combined Algorithm and Hyper-parameter Selection).

While solving a CASH problem will take care of one step in the Machine Learning pipeline, the truth is that there are still a number of steps left unresolved, including pre-processing, feature selection, and model evaluation. ML Pipeline Search aims to solve this issue by automatically

selecting the best pipeline structure for a given Machine Learning problem. The goal is to find a sequence of steps that are able to take a data-set as an input and produce an optimal model as an output.

Like before, it is possible to extend previous methods of Bayesian optimization to solve the Pipeline Search problem. However, as we will see in the following section, methods centered around evolutionary algorithms are proving to be a very good fit for this issue, and researchers have focused on these approaches during the last few years.

2.4 The state of AutoML

After exploring the basics of AutoML in the previous section, we will now go into some more detail about the recent advancements made in this field, as well as how it can best serve us in building a Machine Learning Pipeline.

2.4.1 Novel Approaches

In an article published this year (2022) [39], Pavel Vychuzhanin et al. utilised a novel evolutionary approach to solve Pipeline Search problems. In this article, the authors differentiate between three identified pipeline types, as shown in Figure 2.7. Fixed pipelines, which contain only a sequential array of operations. Variable pipelines, which can contain multiple paths operations, but with uniform length. And lastly, composite pipelines, which, may also contain multiple paths of operations, but these do not require a uniform length.

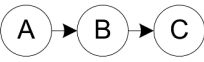
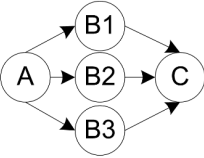
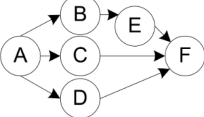
Structure	Graph type	Represented modelling workflow
	Path graph (only one parent for each node)	Linear pipelines
	Minimum spanning graph (all paths between all pairs of nodes have the same length)	Ensemble pipelines (low variability)
	Directed acyclic graph (with all paths finishing in the final node)	Composite pipelines (high variability)

Figure 2.7: Pipeline types defined in [39]

In the AutoML field, most pipeline search solutions are geared towards Fixed pipelines, with only a few trying to solve for variable pipelines. In this article, however, the authors decided to tackle the composite pipeline search problem.

As in explained on the paper, in contrast with the other types, composite pipelines provide the necessary structure to develop highly complex ML systems, with the ability to integrate multiple data sources of differing types, and multiple models, which makes ensemble learning architectures possible.

To tackle this issue, the authors split the problem in two. First they use a genetic algorithm to select a good pipeline structure proposal. Afterwards, Hyper-parameter tuning is performed on the selected pipeline, in order to produce the final result, which is exported as a description of the system in a JSON format.

In order to validate the proposed solution, it was tested on a variety of benchmark problems and compared against the existing state of the art. As can be seen in Figure 2.8, this articles pipeline (FEDOT) managed to beat its competition in 10 different problems (5 regression, and 5 classification tasks), across almost all quality metrics.

In the end, the paper concluded with the uses that this technique could have in Workflow Management Systems, such as, co-designing ML systems architectures, and made the discussed work available as part of their open-source framework (FEDOT).

Quality metric	Pipeline and framework	Regression data sets				
		1027_ESL	1096_FacultySalaries	227_cpu_small	228_elusage	605_fri_c2_250_25
MAE	Static (XGBoost)	0.441±0.010	1.597±0.027	2.006±0.023	10.484±0.037	0.349±0.019
	Composite (FEDOT)	0.430±0.009	1.120±0.012	1.931±0.007	9.451±0.020	0.371±0.033
	Variable (TPOT)	0.432±0.004	1.508±0.023	2.064±0.011	11.051±0.031	0.373±0.014
	Linear (MLBox)	0.434±0.018	3.670±0.048	1.934±0.005	21.075±0.054	0.423±0.076
RMSE	Static (XGBoost)	0.588±0.028	2.199±0.037	2.799±0.023	14.053±0.023	0.448±0.025
	Composite (FEDOT)	0.566±0.013	1.403±0.024	2.759±0.007	11.558±0.020	0.466±0.035
	Variable (TPOT)	0.570±0.009	1.789±0.033	2.886±0.023	14.863±0.032	0.465±0.014
	Linear (MLBox)	0.628±0.052	4.894±0.061	2.793±0.023	26.141±2.27	0.505±0.089
Quality metric	Pipeline and framework	Classification data sets				
		flare	ionosphere	labor	magic	spect
F1	Static (XGBoost)	0.299±0.110	0.939±0.026	0.895±0.023	0.793±0.013	0.837±0.077
	Composite (FEDOT)	0.332±0.057	0.919±0.026	0.931±0.013	0.817±0.004	0.893±0.062
	Variable (TPOT)	0.292±0.034	0.774±0.052	0.840±0.023	0.809±0.004	0.830±0.046
	Linear (MLBox)	0.181±0.097	0.933±0.088	0.840±0.069	0.812±0.017	0.845±0.071
ROC AUC	Static (XGBoost)	0.693±0.025	0.956±0.007	0.923±0.023	0.915±0.015	0.736±0.075
	Composite (FEDOT)	0.708±0.033	0.951±0.011	0.958±0.019	0.930±0.004	0.779±0.043
	Variable (TPOT)	0.701±0.008	0.954±0.079	0.958±0.023	0.928±0.004	0.657±0.039
	Linear (MLBox)	0.509±0.012	0.907±1.036	0.515±0.058	0.856±0.026	0.628±0.031

Figure 2.8: Composite pipeline results as shown in [39]

2.4.2 Performance of AutoML Systems

AutoML is a field of research that promises very exciting results. If attained, it could revolutionize the Machine Learning industry, as well as the entire world. However, current AutoML approaches are still quite distant from the so desired general-purpose Machine Learning tool.

It was shown in [33], by Zöller and Huber that, despite attaining remarkable results, AutoML systems still fall short of expert human performance. In their paper, they started by conducting a small survey on the current state of AutoML research, followed by an extensive analysis of CASH and full pipeline-building AutoML frameworks. They tested 8 different CASH methods in

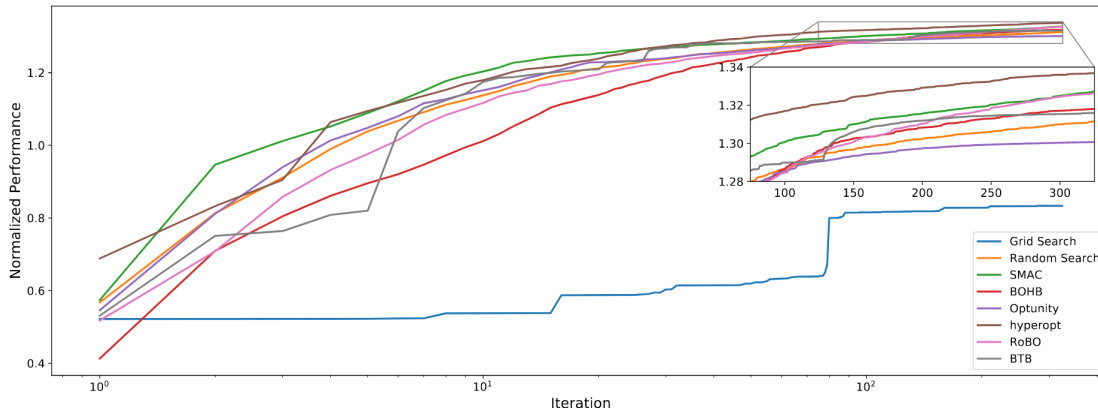


Figure 2.9: Comparison of different Hyper-parameter optimization techniques performed in [33]

114 data-sets, in which they concluded that, aside from Grid-search, which was slightly worse, all algorithms attained similar performance, as can be seen in Figure 2.9. Additionally, 5 full AutoML frameworks were also tested in 73 data-sets.

From this analysis it was possible to conclude that all methods were, on average, better than a simple random forest, however, the results were highly variant and dependant on the data-set. Furthermore, it was concluded that both CASH and AutoML frameworks have a tendency to over-fit on their data, and that CASH methods seemed to perform slightly better than full Auto-ML frameworks, with the draw-back that they can not operate on data-sets with irregularities, such as missing values.

Lastly, a comparison between AutoML and Human frameworks was made with 2 publicly available Kaggle data-sets. In neither experiment was AutoML able to beat the best Humans, with the closest one being the H20 AutoML framework, which was placed on the top 37 percentile on one of the data-sets.

2.4.3 The usefulness of AutoML

Since AutoML is still quite far from becoming the idealized end-to-end general tool that researches hope for, it is important to focus on the gains that it can provide today, to enhance human experts.

In [32], Wang et al. started by synthesizing a human-centered AutoML framework from the existing literature and marketing reports. This framework was composed of 6 Actors, 10 Stages, 43 sub-tasks, 5 levels of Automation and 5 levels of Explanation. With this framework in mind, a survey was conducted on 217 professionals who fit into at least one of the 6 defined Actor roles.

The goal of this survey was to understand current usage of AutoML by professionals, pain points experienced throughout the development process of Machine Learning Systems, and perceived desired levels of automation by the different roles.

The survey suffers from a biased sample size, as all participants belong to the same company, and most of them have only 1-5 years of experience, however, some interesting conclusions were

derived.

The first major conclusion was that ML practitioners desire a higher level of automation than they currently utilise, but not full automation. It is also important to note that each role is more critical of automation in the stages where they participate. With that aside, the stages where higher levels of automation were more consensual were feature engineering, model building, and model deployment.

Another important aspect discussed in the survey was AI explainability, which was deemed important by all participants in most stages of the pipeline, with Expert Data-Scientists considering it extremely important on all stages of development.

Another study which discusses AutoML usefulness is [16]. In this paper, Chauhan et al. shows the multiple steps that AutoML tools attempt to automate, and provides a table comparing the most popular tools in terms of functionality.

Additionally, this paper also portrays a real-world success story about the implementation of AutoML by GNP, a Mexican insurance company. In the given example, GNP faced problems with the development of Machine Learning, as these were too expensive to produce and required a significant number of highly trained Data Scientists. By incorporating AutoML into the development workflow, GNP was able to improve their models performance by as much as 30%, all with minimal human intervention and without the need of employing more Data Scientists.

2.5 State of the Art Conclusions

To answer our first research question, and start solving the design and implementation problem of creating an adequate ML pipeline, it was necessary to conclude our research with the following list of required supported features:

- Data Versioning
- Data Analysis
- Data Cleaning
- Feature Engineering
- Model Training
- Hyper-Parameter Tuning
- Model Evaluation
- Model Versioning
- Experiment Tracking
- Model Deployment

- Model Monitoring

Additionally, since there are many ways of supporting the aforementioned tasks, we have dedicated the following subsections to outlining some of the insights gained from our MLOps and AutoML research. In this subsections, we will further explain what we believe to be proper ML development practices, and how pipelines can help developers to follow them.

2.5.1 Human-Centered Automation

One of the most important aspects of an ML pipeline is how well it automates the Machine Learning development process. For this goal, AutoML algorithms are certainly an enticing option, since they are capable of automating the entire process, from the moment that data is imported to when the model is produced. Some researchers even go beyond this impressive feat, and propose architectures that automate the deployment and re-training steps [12]. However, despite these results, which are coming ever closer to catching up with the quality of an average Data Science practitioner, as was shown in the previous section, we still find them insufficient for a real-world scenario.

As of this date, we believe that the true potential of AutoML does not lie in how well it can replace Data Scientists, but in how well it can enhance their work. That is why a Human-Centered approach was chosen, in detriment of a fully automated pipeline. So, what exactly does a Human-Centered automated pipeline look like?

The key aspect that sets it apart from an automated pipeline is that a Human-Centered pipeline is designed to work in tandem with the developers. It is always interactive, and only becomes ready to move onto the next step when the user gives it the go-ahead, as can be seen in the diagram showed in Figure 2.10.

The interactivity provided by such approaches is crucial, as the developers are the only ones capable of assessing the quality of the data, and the only ones capable of judging whether the results are satisfactory. Furthermore, this system promotes collaboration within the team, as the responsibility of deciding which step to take next lies with them.

For instance, with a fully automated system, if the final model is not able to achieve the desired level of accuracy, the best course of action would be to have a Data Scientist interpret the employed AutoML algorithms, and tweak them in the hopes of improving the performance.

The problem, however, is that this is a very roundabout way of dealing with the issue at hand, which might not even be about the models, but related to the data itself. In which case, it would be better to have a Data Engineer look at the results, instead of a Data Scientist.

In a Human-Centered approach, this kind of interactivity is built-in from the start, and it becomes possible for the members of the team to use their knowledge to improve upon, and direct, what the AutoML systems produce.

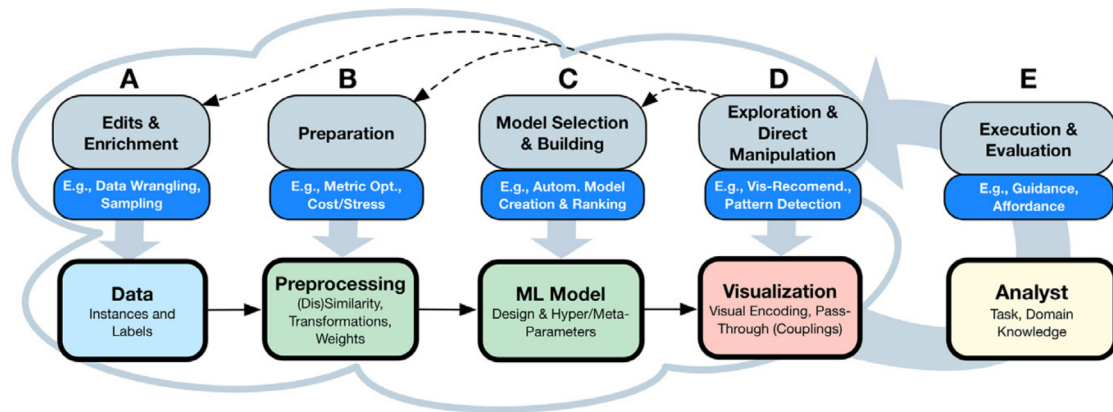


Figure 2.10: Proposed conceptual framework: a reference interactive VA/ML pipeline is shown on the left (A–D), complemented by several interaction options (light blue boxes) and exemplary automated methods to support interaction (dark blue boxes). Interactions derive changes to be observed, interpreted, validated, and refined by the analyst (E). Visual interfaces (D) are the “lens” between ML models and the analyst. Dashed arrows indicate where direct interactions with visualizations must be translated to ML pipeline adaptations. As exposed in [6]

2.5.2 How Machine Learning Pipelines can help with Debt Management

As we saw in this chapter, technical debt can be a hindrance to software development projects, and with the added complexity of Machine Learning systems, it is an even worst problem in this field. There are, however, some things that ML pipelines can do to help manage debt in a responsible way.

As previously explained, technical debt arises from the choice of prioritizing speed over quality. And while at first glance this may seem like a bad thing, in the practical domain, that is not always the case.

The main argument for consciously taking on technical debt, is that arriving to market faster can be more valuable than having a near-perfect solution. In this case, taking on technical debt is just as useful as taking on financial debt, as it allows developers to do things at a pace they otherwise would be unable to.

With that said, there are some dangers when handling technical debt. The main concern is that when a team is developing with a speed-first approach, it is easier to unknowingly introduce debt into the project. Unknown debt sources are extremely detrimental to the software development process, as they can silently pile up over time, making it more and more difficult to keep developing without the need for major reworks on the underlying structure.

Another issue to keep in mind is that, even when the decision to acquire technical debt is conscious, it is not always immediately clear what its exact cost will be. It may be the case that the true cost of a given debt source only becomes apparent much later on in the projects life-cycle, as such, debt should always be addressed as soon as possible.

All in all, technical debt is an unavoidable aspect of Machine Learning development, and of any software development project for that matter. However, ML pipelines can help manage this debt by taking care of some easier to deal with debt sources from the start, thus reducing the number of potential issues that have to be kept in mind [31]. Some of these sources are:

- Undeclared Consumers Debt
- Glue Code Debt
- Data Testing Debt
- Monitoring and Testing Debt

Pipelines can help with these issues thorough several methods. By automating testing and monitoring tasks, and producing alerts to warn the developers, it is possible to tackle the last two debt sources. Where as, by keeping track of resource access and enforcing the use of tools that minimize glue code, such as ML Bazaar [60], the first two sources can be minimized. In our proposed pipeline we will attempt to address some of these issues ourselves.

2.5.3 Data-Centric vs. Model-Centric

Unlike traditional software, an ML project's quality is not just dependant on code. When attempting to improve the results of a given Machine Learning system, the data and model components play a crucial role in the decision making process. Ant any given point, it is necessary to choose whether it is most appropriate to tweak the code that generates the models, or if it is best to change the data itself. In Table 2.1 we show a comparison between the two methods.

Model-Centric ML	Data-Centric ML
Working on code is the main objective.	Working on data is the main objective.
Focus on model optimization, so it can deal with noisy data-sets.	Focus on improving the data quality, and removing excess noise.
Data labels may be inconsistent.	Data consistency is key.
After standard processing, data is fixed,	Code / algorithms are fixed.
Model is iteratively improved.	Data quality is iteratively improved.

Table 2.1: Comparison between Model-Centric and Data-Centric approaches derived from [19]

The Data-Centric approach to Machine Learning development states that, in most cases, assuring the quality of the data is more important than changing and optimizing the models. This is best described by the well-known expression "Garbage in, garbage out", which promotes the idea that regardless of how sophisticated our models are, if the data we are feeding them is sub-par, so will our results.

In contrast, Model-Centric Machine Learning development prefers to iterate through increasingly complex models, and optimize hyper-parameters, in hopes that the algorithms will be robust enough to learn adequately, even with relatively noisy data-sets [19].

In a practical setting, there are good arguments for both approaches, with no clear winner without further domain-specific context. Ideally, we would be able to acquire high quality data and then choose a sufficiently complex model to learn from it, however, that is not always possible.

The Data-Centric approach usually requires developers to re-collect, re-label and re-analyse data iteratively, until a certain data quality metric is achieved. However, sometimes it is not feasible, or cost-efficient, to re-collect large amounts of data that many times. Not to mention that defining data quality metrics is an uncertain art by itself, as there are few widely agreed-upon standards, given the many forms that data can take.

Lastly, some problems really do require complex Machine Learning models, regardless of how good our data is. For example, if we collect data that perfectly describes a quadratic function with 0 noise, but then attempt to apply a linear regression model, we will never get good enough results.

Despite these challenges, it is still difficult to argue that one should not try to improve its data collection, or data engineering. It might be difficult to do so in some cases, but it should always be attempted before resorting to higher complexity models. And the truth is, that despite the evidence of significant gains from employing a Data-Centric approach, as of March 2021, and according to sources such as Stanfords University professor, Andrew Ng, more than 90% of research conducted in the Machine Learning field is Model-Centric [21].

Due to this imbalance in practitioners inclination to Model-Centric approaches, we find it extremely important to incentivize Data-Centric development through the Machine Learning pipeline, and will provide the necessary tools to do so in our proposal.

2.5.4 The need for Lightweight Solutions

To finalize our conclusions from the state of the art research, we must discuss one of the most silent problems of Machine Learning pipelines. The fact that they are still ignored by many ML practitioners.

As we saw on the survey conducted in [26], a large portion of Machine Learning developers find themselves working on the initial stages of ML, usually with small-scale proof of concept projects. Due to the experimental aspect of these projects, and the small amount of available resources, concerns with infrastructure are often left unaddressed, as they are not crucial for that stage. But there are two problems with this.

First, if the project does progress into a serious implementation, it will be necessary to adopt a pipeline at some point, and the process of migrating everything to said pipeline can be quite cumbersome. For example, it could require versioning, organizing and annotating a bunch of data-sets and models with the proper metadata, transforming several jupyter notebooks into isolated pipeline steps on a DAG, standardizing coding styles and nomenclatures, creating documentation for the entire project, changing deployment strategies, etc.

Aside from the error-prone work of performing this exhausting migration, which is in itself a late technical debt payment, there is also another argument. As we saw before, by enforcing proper development practices, Machine Learning pipelines contribute to the overall quality of our ML systems. As such, aside from freeing the team of the migration pain, having a pipeline from the start can also help in making the initial experiments more successful! So, why is it that so many practitioners decide to forgo these benefits?

As of today, there is not an unquestionable answer, but referring again to the survey in [26], we can infer from the respondents situations that it is both due to time pressures and lack of resources. After all, if these small Machine Learning teams want to adopt a pipeline, they really only have two options. Build their own, or integrate a third party solution.

Building a proprietary pipeline from scratch can be very time consuming. It is necessary to analyse the entire ML process, decide on a desired workflow, compare between all the tools and libraries available, choose a tech stack, and then actually build and maintain the pipeline. For a small team who is already low on resources and time, dedicating even just one member to this endeavour might seem too costly, even when it is the best choice.

Using a third party solution can seem appealing in these cases, however, it has three drawbacks. First, most end-to-end ML pipelines are commercial products, which might strain the budget of some projects, even if most do provide a free trial license. Secondly, there is an inherent risk when choosing an external pipeline. The team might go through the effort of learning how it works, only for it to prove itself inadequate later down the line. And with no way of adapting third party pipelines to fit the teams evolving needs, the only option left is to migrate everything into a new, more adequate, pipeline, which can pose many of the same headaches as before. Lastly, even with open-source solutions such as Kubeflow [54], which is free and more easily extendable, there is the problem of unnecessary complexity. After all, it is not every project that requires a Kubernetes backed working environment hosted on the cloud, specially with the effort and resources required to set it up.

Now, while the previously mentioned solutions do have their drawbacks, it is worth noting that they are still formidable tools, and for many projects they will still be the best option. However, for the equally large number of projects that need a different solution, we propose to invest on truly lightweight pipelines, both in computing resource consumption as well as setup overhead.

On the following chapters we will go over how our proposal fits these criteria, and how it helps accomplish our goal of widening the adoption of Machine Learning pipelines.

Chapter 3

Our Approach

In this chapter, we explain our proposed solution to the first research question exposed in Section 1.3. To understand our approach for this problem it is necessary to keep in mind the shortcomings identified in the state of the art. Most crucially, the lack of extensibility in most third-party products, and the demanding setup overhead these solutions require, both in resources and time.

Starting with Section 3.1, we showcase a representation for Machine Learning workflows, and the need for such an abstraction in the context of this dissertation. Next, in Section 3.2, the decisions behind the proposed architecture are explored, and an argument is put forward, in regards to why we chose to separate the pipeline problem into two tools, a framework entitled Quick Machine Learning (QML), and the actual pipeline (Lightweight Pipeline). In Section 3.3, we delve deeper into the features of QML, and what it enables us to do. In Section 3.4, the architecture for the Lightweight pipeline, and its workflow is explained. And lastly, in Section 3.5, we demonstrate our tools with the help of a small Machine Learning project, developed with the Lightweight Pipeline, running on QML.

3.1 Chosen Representation for Machine Learning Pipelines

The term "Pipeline" can sometimes generate some confusion in the Machine Learning field, as it can refer to the sequence of algorithms used to turn a data-set into a predictive model, as is the case in papers such as [39], [33], and [47]. However, in this dissertation the term is extended to encompass other phases of the Machine Learning workflow, as is done in [17].

Before discussing the particularities of the proposed pipeline, it is first necessary to understand the anatomy of a typical pipeline-driven Machine Learning workflow. Unfortunately, not all workflows are exactly the same, and authors provide different representations for their structure, as such, a comprehensive diagram for the general Machine Learning workflow adopted in this dissertation is proposed below. This diagram is a compilation and reorganization of structures presented in [32], [30], and [28], and was built as a flowchart based on a simplified version of the open standard for Business Process Modeling Notation (BPMN), as can be seen in Figure 3.1.

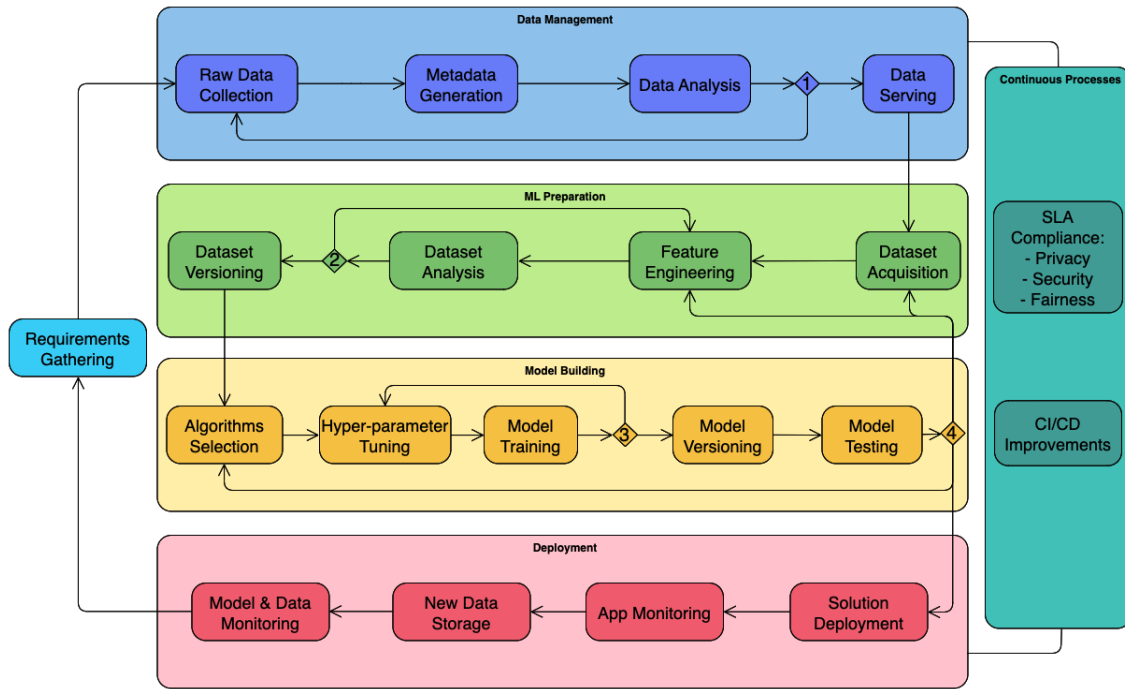


Figure 3.1: General Machine Learning Workflow derived from [32], [30], and [28]

Just like in [30], our proposed workflow contains 4 distinct phases (Data Management, ML Preparation, Model Building, and Deployment), as well as a Requirements Gathering step, which lies outside the 4 phases.

Additionally, an extra "Continuous Processes Phase" is portrayed. Unlike the previous phases, this one does not appear in sequence to any other, instead, it is used to include all tasks that must be continuously executed throughout the development of the Machine Learning system. Most commonly, these steps are related to SLA compliance, and improvement of the pipeline itself.

Aside from the aforementioned extra Phase, we also propose another two alterations to previous depictions of the Machine Learning Workflow.

First, the cyclical aspect of the workflow is made explicit by connecting the end of the Deployment Phase with the Requirements Gathering step. This way, it is possible to show that, throughout the life-cycle of a Machine Learning project, requirements may change, as is the case when phenomena such as concept or data drift occur [28].

Secondly, 4 decision nodes are added throughout the course of the diagram's flow. These nodes serve to enforce the notion that developing Machine Learning systems is an iterative process, and, as such, it is expected to go back to previous steps and try out different approaches.

Lastly, it is important to note that several stages of the workflow may be simultaneously active, which is due to the collaborative and continuous nature of Machine Learning projects. As an example, at the end of the Model Testing step, it is possible for a team to continue the project into the Deployment Phase (with the tested model), but, at the same time, a different team can also go back into the ML Preparation Phase, in order to develop a better module for a future service

update.

As a closing remark, this dissertation will not go in depth on the details of each individual pipeline step, as they are already thoroughly discussed in the cited literature. As such, if further insights into the specifics of these tasks is desired, please refer to articles [32], [30], and [28].

3.2 Quick Machine Learning Framework Architecture

In an attempt to provide an adequate solution to the first research question, we began by designing a lightweight pipeline capable of handling most of the steps in our workflow diagram (Fig. 3.1). Unfortunately, we soon realised that simply creating an open-source pipeline, even if well designed, did not provide the required amount of extensibility. In fact, had we kept this approach, it would, at most, have ended up as extensible as something like Kubeflow, which would not have been satisfactory for our purposes.

Instead, we decided to split the problem in two. The first part consisted of creating a framework which could run any specified pipeline, as long as it followed a certain representation schema. The second part involved creating the actual pipeline, and representing it through a human and machine readable file. This framework was then entitled Quick Machine Learning (QML), and the developers can use it to interact with any supported pipeline, as can be seen in Figure 3.2

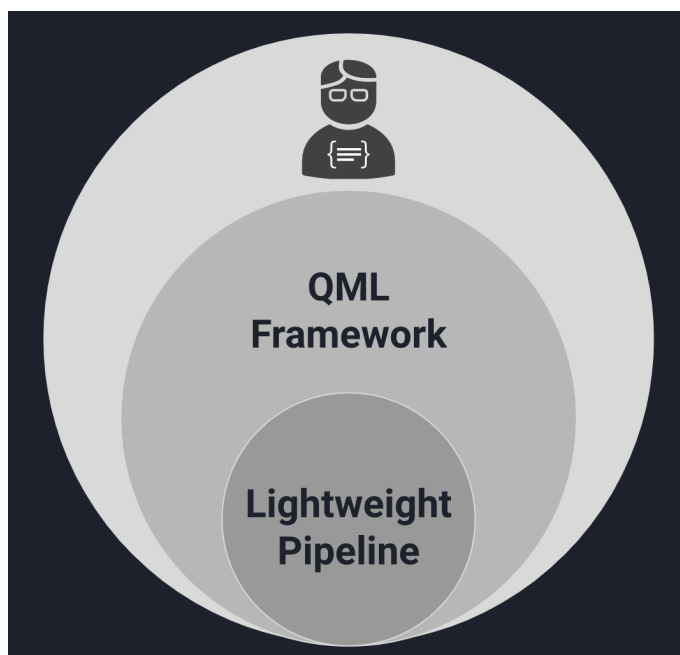


Figure 3.2: Architecture of our proposed solution. At the top, the developer interacts with QML, which then utilises the selected pipeline, in this case, the Lightweight Pipeline.

The main advantage of utilizing such an architecture, is that it becomes possible to modify a pipeline simply by changing the contents on the specification file. As such, if the developers want

to add a certain module to their pipeline, they only need to reference it on that file. Analogously, they can also remove features that way.

Additionally, this architecture enables different projects to utilise different pipelines through the same interface, which minimizes the learning curve when switching from one pipeline to another.

3.3 QML Framework capabilities

The QML framework is a CLI tool built with python, which provides the user with only 2 commands. Start and Edit.

The 'start' command is the most important one, as it enables the users to create a new development environment with any chosen pipeline, or, alternatively, to boot up any previously created environment.

The 'edit' command, on the other hand, simply outputs the directory where the existing pipelines are installed, so that new pipelines can be added, or so that existing ones can be edited, deleted, or updated. Additionally, all pipelines installed in this directory must follow the same structure as the one represented in Listing 3.1.

```

1 example_pipeline_name/
2 |— .env.yaml
3 |— assets/
4 |   |— file_1.txt
5 |   |— file_2.jpg
6 |— modules/
7 |   |— cli_command_1.py
8 |   |— cli_command_2.py
9 |   |— cli_command_3.py
10 |   |— event_handler_1.py
11 |   |— event_handler_2.py
12 |   |— general_event_handler.py
13 |   |— setup_process_1.py
14 |   |— setup_process_2.py

```

Listing 3.1: Example of an installed QML pipeline directory structure.

At the root of the installation, the folders name will represent the name of the pipeline. Then, immediately bellow it, the specification file must be saved in a YAML format, and entitled '.env.yaml'. Lastly, two more directories can be included. A 'modules' directory, where all of the python modules utilised by the pipeline are saved. And an assets directory, for data files such as a 'requirements.txt', or any other file type the pipeline might need to access.

As previously stated, to run the pipeline, QML reads the '.env.yaml' specification file, and executes its processes accordingly. In Listing 3.2, we have included an example of a typical pipeline specification file, where its four main sections are showcased:

```
1 version: 1.0.0
2 setup:
3   structure:
4     - folder_1:
5     - folder_2:
6       - subfolder_1:
7         - file_2.jpg
8       - subfolder_2:
9         - file_1.txt
10  processes:
11    - setup_process_1
12    - setup_process_2
13 watchdog:
14   - directory: folder_1
15     events:
16       on_created:
17         - event_handler_1
18         - event_handler_2
19       on_modified:
20         - event_handler_2
21       on_deleted:
22       on_moved:
23       on_any_event:
24         - general_event_handler
25   - directory: folder_2/subfolder_1
26     events:
27       on_any_event:
28         - general_event_handler
29 commands:
30   - command_name: cli_command_1
31   - command_name: cli_command_2
32   settings:
33     ignore_unknown_options: True
34     allow_extra_args: True
35   - command_name: cli_command_3
```

Listing 3.2: Example of a specification file used by QML to describe a pipeline.

- Version
- Setup
- Watchdog
- Commands

The 'version' keyword serves merely to identify the version of the pipeline. This is checked to prevent a user from utilizing an inappropriate pipeline version that might be installed in their system. If the required and installed versions do not match, the user is prompted to update their installed pipeline in order to match with the projects requirements. The other three sections, however, provide more intricate functionality.

3.3.1 QML Pipeline Initialization

The 'setup' keyword identifies the section that contains the necessary information to setup the project environment properly. Here it is possible to specify two things: A directory structure that must be complied with throughout the development of any project running this pipeline. And, a list of processes that are executed whenever the environment is initialized.

To understand how this section operates, as well as how the following sections are integrated into QML, it is first necessary to understand how the 'start' command functions. The 'start' commands purpose is to initialize a pipeline environment, whether that entails creating a new one for a new project, or simply booting up an existing environment. For that, the user can call 'start' with three options, as exemplified in Listing 3.3

```

1 Usage qml start [OPTIONS]
2
3 Options
4 -p, --path TEXT      Name of root project directory relative to current
5                      directory
6 -conf, --config TEXT Name of configuration file for setup of a new
7                      environment
8 -v, --version TEXT   Python version to ask for if building a new
9                      environment

```

Listing 3.3: QML's "start" command options.

The 'path' option gives QML the path where the environment must be initialized, and defaults to the current working directory if empty. The 'config' option is used only when creating a new environment, and tells QML what pipeline to initialize. If empty, the 'config' option will default to the Lightweight Pipeline. The 'version' option is also only meant for new environments, and it is used to enforce a specific python version for the project. If empty, no version will be enforced.

With this information, QML checks whether the provided path already contains an existing environment, or if a new one must be created. The steps for creating or initializing an environment are very similar, for the most part.

Both processes start by reading the YAML 'config' file, and generating the provided folder structure, as well as copying any files from the 'assets' folder if they are referenced, as specified in Listing 3.2.

Then, both check whether a virtual environment for the project has already been created or not, and in case there is none, a new one is constructed. Additionally, if a required python version was specified, and said version is installed on the users machine, it is used to build said virtual environment.

After ascertaining that a virtual environment exists, QML executes the 'processes' in the specification file (Listing 3.2) in the order that they appear. These processes are simply python files included in the 'modules' directory (Listing 3.1), which contain a 'runProcess' function, as shown in Listing 3.4. These functions can also accept a list of strings as an argument, so that the users can call the 'start' command with custom options accepted by the specific pipeline. For example, the Lightweight Pipeline can be called with an option '-t' which, if enabled, includes a template project in the generated directory.

```
1 def runProcess(args : "list[str]"):
2     """This is a setup function that will be called during the pipeline
        initialization process."""
3     pass
```

Listing 3.4: Example of a setup process.

Lastly, after executing all the setup processes, QML activates the virtual environment, switching the users terminal shell, and proceeds to the 'watchdog' section.

3.3.2 QML Watchdog

The 'watchdog' keyword is used to define a set of directories that the pipeline must watch. As shown in Listing 3.2, for each of these directories a set of events can then be specified:

- Creation (on_created)
- Modification (on_modified)
- Deletion (on_deleted)
- Movement (on_moved)
- Any (on_any_event)

When the specified events are triggered on the specified directory, the pipeline will call the designated event handlers, in the order in which they are referenced. Once again, these event

handlers are merely python files that implement a 'runEvent' function, as shown in Listing 3.5. This function takes as an argument a 'FileSystemEvent' object, from python's watchdog library [42], which can then be used to access additional information about the event, such as created file name and moved location. These event handlers can then be used to automate a lot of internal pipeline processes, as we will show in our Lightweight Pipeline implementation.

```
1 def runEvent(event):  
2     """This is an example of a handler that will be called when a file triggers an  
   event."""  
3     pass
```

Listing 3.5: Example of an event handler file.

3.3.3 QML Commands

When running outside the context of a pipeline environment, QML only has the two previously mentioned commands. However, after the virtual environment is activated, the 'start' command disappears. Instead, QML adds to itself all of the specified commands, under the 'commands' section, as is shown in the Listing 3.2.

Once again, each of these commands corresponds to a python file in the 'modules' folder, with the same name. All of these files must implement a 'runCommand' function, and it is possible to extend its functionality with annotations from the 'click' python package [70], just as demonstrated in Listing 3.6. With click, the pipeline creators can easily add typed arguments, options, input prompts, etc. And through the YAML file they are also able to modify the behaviour of the click commands with the use of an optional 'settings' keyword, as is demonstrated on the 'cli_command_2' (Listing 3.2), which enables the command to accept any number of unknown options when it is called.

```
1 import click  
2  
3 @click.argument('arg', type=str, default='default_value')  
4 def runCommand(arg):  
5     """This is a template for a CLI command available in the pipeline."""  
6     pass
```

Listing 3.6: Example of a command file from our pipeline.

Utilizing these CLI commands in the environment is a great way to enable quick interaction between the developer and the pipeline, and is used to great effect in our Lightweight Pipeline implementation.

3.3.4 Creating vs. Initiating an Environment

As previously explained, after calling QML's 'start' command, the resulting environment behaves in exactly the same way, regardless of whether we are initiating an old environment, or creating a new one. However, the main difference behind these two processes is that, after a new environment is created, a file entitled '.qml_env.yaml' is automatically added to the root of the project.

This is a simple file, whose only function is to signal that an environment has already been created, and save the information necessary to run it. In this file, the name of the projects pipeline is saved under the 'name' key, the pipelines version is saved under 'version', and the python version required for the project is saved under 'python_version'.

With this small inclusion, it is possible to come back to an existing project without re-running the entire creation process. Additionally, the user can also re-create a pipeline environment after cloning a project for the first time, by simply calling 'qml start'. This way it is possible to ensure that everyone on the development team can reproduce the project on the same pipeline, and with the same python version, running on equivalent virtual environments.

3.4 Lightweight Pipeline Architecture

In order to take advantage of QMLs features, it is necessary to provide it with a compatible pipeline. As such, we set about designing the architecture for the Lightweight Pipeline, so that it could be implemented within QML.

For this task, we followed the workflow diagram proposed in Figure 3.1. By basing ourselves on that schema we were able to develop a comprehensive architecture that takes into account the entirety of the Machine Learning workflow, as can be seen in Figure 3.3.

To understand this diagram, it is first necessary to talk about the meaning behind some of our visual elements. The first thing to note is the inclusion of gray nodes. These gray nodes represent miscellaneous steps, which sit outside the aforementioned tasks in Section 3.1, but that are necessary to operate the pipeline, even though they do not contribute directly to any ML specific task. In our case, the only "waste" steps are the first two, which are required to setup the development environment.

Another glaring difference between the original workflow diagram and our pipeline diagram is the inclusion of dotted elements. These dotted elements represent tasks that our pipeline does not yet support. For example, as this pipeline is supposed to be domain agnostic, we have decided to forgo the "Raw Data Collection" step entirely. Instead, it is assumed that the developer already has a data file ready to be utilized.

Lastly, we differentiate between manual steps, fully automated steps, and hybrid steps. The manual steps are represented with simple rectangular boxes. The hybrid steps add a small circle at the top of the rectangular boxes. And the fully automated steps have rounded edges, as well as the circle on top.

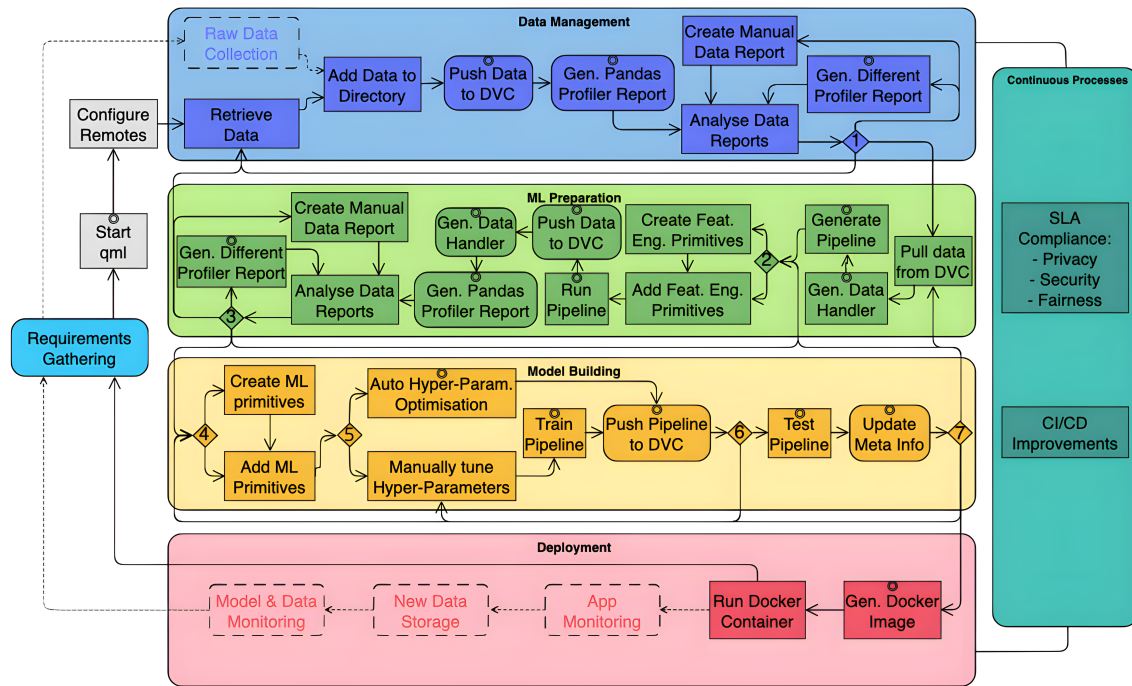


Figure 3.3: Diagram of our proposed Lightweight Pipeline.

To check our implementation of QML and the Lightweight Pipeline, please consult the repository at <https://github.com/WALEX2000/qml>.

3.4.1 Data Management Phase

Data is a necessary commodity for any Machine Learning project, as such, every project starts in the Data Management phase, which follows the steps described in Figure 3.4.

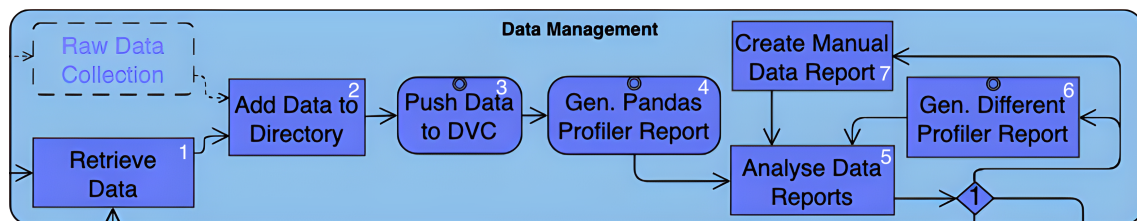


Figure 3.4: Enlarged diagram of the Data Management phase, in the Lightweight Pipeline.

The first step, is to manually retrieve a data file for our problem, and then add it to the appropriate folder in the project directory (step 2). Automatically, our pipeline pushes the designated file to DVC (step 3), a data version control tool that offloads the versioning responsibility from git, due to the large expected size of data files. Additionally, an html report on the data-set is automatically generated, with the use of Pandas Profiler (step 4). Pandas Profiler is a data-set analysis tool, which assesses data quality according to standard metrics, and produces useful visualizations.

Lastly, the developers must analyze the generated report, and make their first decision (decision node 1). Here, they can choose to follow in four different directions:

Decide that the data file's quality is insufficient, and go back to the first step of this phase (look for a different data-set). Decide that a deeper analysis is required, and call Pandas Profiler through our CLI, and generate a more comprehensive report on the data (step 6). Or, in case that proves insufficient as well, manually analyse the data in a jupyter notebook (step 7). Lastly, and likely after many iterations of the previous options, the developer can decide that the data is sufficiently adequate, or that no better alternative exists, and proceed to the ML Preparation Phase.

3.4.2 ML Preparation Phase

In the ML Preparation Phase (Figure 3.5, the first step is to pull the desired version of a given data-set from DVC, by running a simple console command. This way, it can be guarantee that the developer is working with the intended files. Next, with the help of on of our CLI commands, the user must generate a data handler, which helps define the meta information associated with a given file, such as train/test splits, target variables, and scoring methods (step 2).

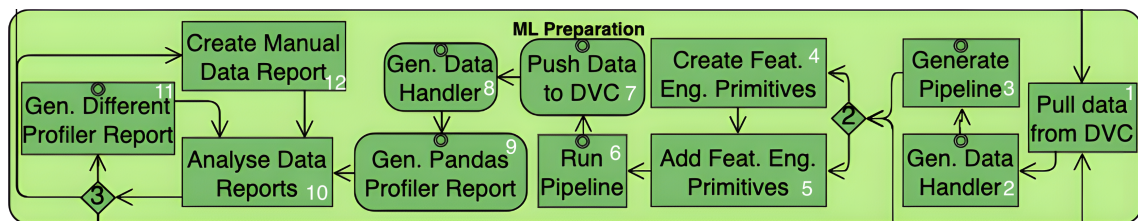


Figure 3.5: Enlarged diagram of the ML Preparation phase, in the Lightweight Pipeline.

After the data-specific tasks are taken care of, the developer can proceed to work on the feature engineering pipeline. To start this process, it is necessary to run a CLI command, which automatically generates a template pipeline (step 3). Then, in decision node 2, the user must make his first choice. If the Machine Learning primitives required are already available in MLBazaar [60], he can go straight to step 5, and simply add them to his pipeline. Alternatively, if the developer requires something too specific, he must create it himself (step 4), and then add it.

After each iteration, the developer should run the pipeline with the appropriate CLI command (step 6), which automatically saves the resulting output, versions it with DVC (step 7), generates the new data handler (step 8), and produces a new Profiler report (step 9). Then, just as in the Data Management phase, the user can analyse the generated report (step 10), and choose between a number of options: (decision node 3)

He can decide that the data-set still needs further improvements, and go back to decision node 2, to keep working on the feature engineering pipeline. Alternatively, he can decide that a deeper analysis is required, and opt to either run Pandas Profiler again (step 11), or to manually analyse the data-set (step 12). Lastly, after a number of iterative improvements to the feature engineering pipeline, the user can decide whether the results are satisfactory, and proceed to the

Model Building phase. Or, if the results are not good enough, but no further improvements can be made, the developer can go back to the Data Management phase and start everything over with a new data-set.

3.4.3 Model Building Phase

In the Model Building phase, the developer can decide to keep working on the previous feature engineering pipeline, effectively turning it into a general pipeline with all the Machine Learning steps. Or, he can create a new pipeline, just as was done in the previous phase. (We omit this last option in Figure 3.6 for simplicity's sake).

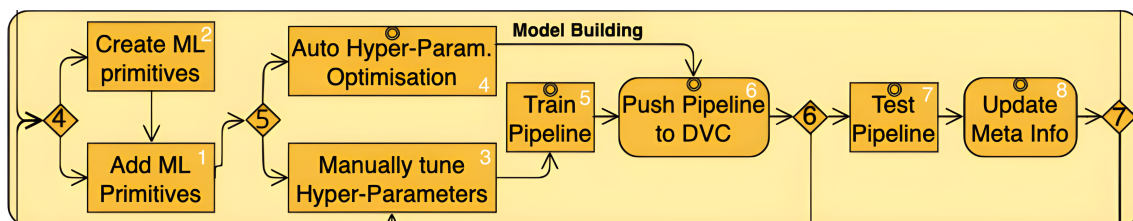


Figure 3.6: Enlarged diagram of the Model Building phase, in the Lightweight Pipeline.

Regardless of the picked approach, in decision node 4, the developer must choose between adding a standard primitive to the pipeline (step 1), or developing a custom one (step 2). Afterwards, in decision node 5, he can decide between manually setting the hyper-parameters (step 3), or running our auto-tuner through a CLI command (step 4), which also automatically trains the model. In case the hyper-parameters are set manually, a CLI command must also be run to train the models (step 5), which will then be versioned in DVC (step 6). At this point the developer is faced with yet another choice (decision node 6). If the validation score is not good enough, he can choose to continue tuning the hyper-parameters, or to go back to decision node 4 and work on the pipeline. Alternatively, in case the results are satisfactory, he can run the test-set on the pipeline through one of our CLI commands (step 7), which also automatically updates the models meta information with the results (step 8).

Lastly, based on the test score, the user must ponder over the last decision in our pipeline (decision node 7). If the developer deems the results to be adequate, he can proceed to the Deployment phase. However, if further improvements are required, the developer can either continue to work on the ML pipeline through hyper-parameter tuning, or changes to the algorithm selections. Or, he can also go back to the ML Preparation phase and change the feature engineering pipeline.

3.4.4 Deployment Phase

When the developers reach the Deployment phase (Figure 3.7), they only need to run our CLI 'deploy' command (step 1). This command generates a docker image for the chosen Machine Learning model with a simple REST API, which provides a Post route that performs a single prediction on a new data entry. After the image has been successfully generated, the user can

decide to host it locally, or in any other docker compatible machine, by running a container from our image (step 2).

As can be seen in our diagram, the last three steps of the Machine Learning workflow are not yet supported by our Lightweight Pipeline. This was not an intended approach, however, due to time restraints during the development of this project we were unable to implement these last few steps. Effectively leaving the deployed system without monitoring.

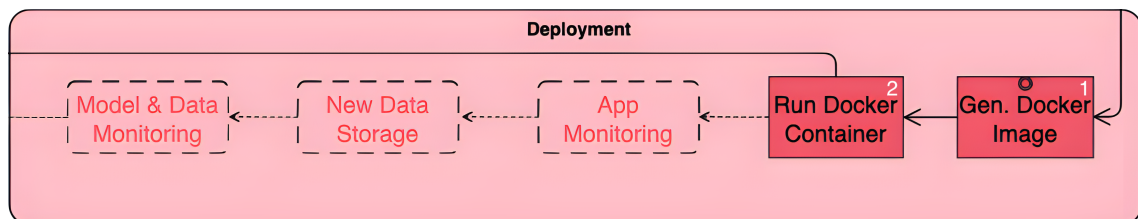


Figure 3.7: Enlarged diagram of the Deployment phase, in the Lightweight Pipeline.

3.5 Lightweight Pipeline Demonstration

In order to test and demonstrate the use of the developed tools, we decided to develop an experimental project with the Lightweight Pipeline, which can be found at: https://github.com/WALEX2000/Wine_Proj. For this demonstration, we opted to utilize the Red Wine Quality data-set [1], due to its simplicity, usability, and popularity. This data-set consists of 1599 samples of Portuguese red wine, which have been evaluated and ranked by three wine tasters, whose median score was recorded as the target variable for this data-set. Additionally, each row also contains information about several wine quality metrics, such as pH level, alcohol content, acidity, etc. As for the chosen task, we will attempt to classify the quality of the wines, given their specifications. As the classes are ordered, we may treat this as an ordinal regression problem.

To start developing the project we created a folder entitled "Wine_Proj", and then ran the "qml start" command, which, by default, utilizes the Lightweight Pipeline. After executing, the command had created had a virtual environment in which to work on, a folder structure as specified in the YAML file 3.2, and a project initialized with Git and DVC. Afterwards, our local Git instance was manually configured with a GitHub remote, and the DVC instance with a Google Drive remote, as specified in their documentation [67]. With these steps concluded, we were able to proceed into the Data Management Phase.

3.5.1 Data Management Phase

The first step of the Data Management Phase was to download the data file from Kaggle [66], unzip it, and add it to the data folder. This triggered the first automated processes of our pipeline, which automatically versioned the file with DVC, and generated a data report. In order to keep the development steps well structured, these changes were committed and pushed Git and DVC.

Immediately after, we ran the command `"qml inspect_data data/winequality-red.csv"`, in order to display the aforementioned data report inside the browser (Examples of these reports can be found in the Annex [A.1.1](#)). The first thing that made itself apparent was the single warning for a high number of zeros in the "citric acid" feature, however, we quickly discarded the issue, as there was no further indication of it representing a zero-inflation problem. Next, we observed the target variable "quality", and found that despite the given 1 to 10 scale it only had values from 3 to 8, and most of them were between 5 to 7, which meant that this was a heavily unbalanced data-set.

Since we wanted to conduct a deeper analysis than this report allowed, we crated a new one with the command `"qml inspect_data data/winequality-red.csv -f"`, which generates a fuller report on the data. This time there were 29 new alerts, most pertaining to high correlation between features, and one to duplicate rows, as such, we set about investigating these issues.

For the duplicated rows, the report simply revealed that there were 220 wines with the exact same characteristics as a previous entry, and with the exact same quality evaluation. As we did not have access to domain knowledge on how likely it is for different wines to exhibit the exact same properties, this raised some concerns. In order to make sure that this behaviour was not due to accidental duplicated entries, we manually created a jupyter notebook in order to perform some more tests.

In this notebook we counted the number of duplicate entries, but this time without taking into account the quality field. With this analysis, 240 wines with the same characteristics as a previous entry we found, which was 20 more than before. This meant that there were 20 wines with differing quality evaluations but matching properties, which let us conclude two things. First, the duplicate rows do not seem to have been created due to errors in the collection process. Secondly, the recorded properties alone can not perfectly predict the perceived quality of wine, as matching properties can have differing results.

In regards to the correlated features, we analysed Spearman's and Pearson's correlation matrices drawn in our report. From this analysis it was concluded that the most correlated variables (higher than absolute 0.4 in either metric) were:

- pH with fixed acidity
- total sulfur dioxide with free sulfur dioxide
- total sulfur dioxide with alcohol
- total sulfur dioxide with residual sugar
- total sulfur dioxide with density
- density with residual sugar
- density with chlorides

- density with alcohol
- alcohol with residual sugar

This means that "total sulfur dioxide" and "density" are both highly correlated with 4 features, "alcohol" and "residual sugar" with 3, and "ph" and "fixed acidity" with each other. Without further domain knowledge it is not possible to say with certainty if any of these metrics are redundant, but it might be wise to drop some features. As such, we committed the changes, and concluded the first phase.

3.5.2 ML Preparation Phase

Moving into the ML Preparation phase, we skipped the first manual step, as the required data was already on our machine, and instead ran the `"qml gen_handler data/winequality-red.csv"` command. This command generates a jupyter notebook where it is possible to define a target variable, which in our case was "quality", as well as a scoring function, for which we chose the F1, RMSE and MSE scores, and the train / test split ratio, where we left it as default (25% for the test set). Then, we ran the `"qml gen_pipeline Wine_FE"`, which creates a jupyter notebook where we can develop the Machine Learning solution for the problem.

Due to the highly imbalanced data-set, and the inability to acquire new data, we decided it was best to shift the classification scale from 1 - 10 to 0 - 2. In this new scale, 0 would represent poor wines with ratings of 1,2,3 and 4. 1 would represent average wines with ratings of 5 and 6. And 2 would represent good wines with ratings of 7 and above. To do this, we created a custom feature engineering primitive, just as specified in MLBazaars documentation [44], which was entitled "re_rate", and then added it to our notebook. After running the command `"qml run_pipeline Wine_FE data/winequality-red.csv -fd -sd winequality-red-FE"`, and subsequently inspecting the new data file, it was confirmed that we had indeed attained the desired results, even if the data was still unbalanced.

Next, we decided to drop features "pH" and "free sulfur dioxide" due to their low impact on quality and high correlation with other features.

3.5.3 Model Building Phase

In the Model Building Phase we attempted to train several models, in order to find the best results. First, we created a new pipeline entitled "Wine_Log" and added a Logistic Regression primitive to it. Then, we trained this model with the command `"qml run_pipeline Wine_Log data/winequality-red-FE.csv -sp"`, and tested it with `"qml test_pipeline Wine_Log data/winequality-red-FE.csv"`, which got us an accuracy of 83%. In order to improve this value we utilised our automated hyper-parameter tuner with the command `"qml run_pipeline Wine_Log data/winequality-red-FE.csv -sp -at 200"` which marginally bumped up the accuracy to 84.5%.

In order to find better results we tried several models, such as Naive Bayes Classifier, XGBoost Classifier, keras's MLP MultiClass Classifier, and a Random Forest Classifier. Since the Random Forest got the best results, we attempted to improve upon it by creating a Random Forest Regressor, in the hopes of keeping some of the ordering information. Due to this Regressor outputting decimal values, we had to create a custom Machine Learning primitive to round out the predictions, however, these results were still inferior to the Random Forest Classifier, as can be seen on the Table 3.1. Since we saw no clear way to further improve the predictions, we decided to deploy our Random Forest Classifier model, with all the steps represented in Figure 3.8.

Model	RMSE	MSE	Weighted F1	Accuracy
Logistic Regression	0.41	0.17	0.72	0.83
Logistic Regression Optimised	0.39	0.15	0.78	0.85
XGBoost	0.39	0.15	0.83	0.85
Naive Bayes	0.44	0.19	0.76	0.81
MLP	0.42	0.18	0.75	0.83
Random Forest Classifier	0.35	0.12	0.87	0.88
Random Forest Regressor	0.37	0.14	0.84	0.86

Table 3.1: Table with Model Scores.

3.5.4 Deployment Phase

In order to deploy our Random Forest model, we booted up Docker and ran the command `qml deploy Wine_RF`. This command created the required docker image with the name `"wine_rf_image"`, which was then launched with the command `"docker run -d --name wine_rf_container -p 80:80 wine_rf_image"`.

In order to test if the deployment had been successful we then used Postman to send a Post request to the endpoint, and waited for the successful reply!

3.5.5 Implementation Conclusion

In this chapter we have addressed our second research question, and it was indeed possible to implement a lightweight pipeline capable of addressing almost all of the required features presented at the end of chapter 2. The only missing ones being the model monitoring, and experiment tracking features, which we were unable to add in time.

Furthermore, with the Wine Quality demonstration, we proved that the Lightweight Pipeline can be used to solve standard, tabular classification / regression problems. And while validation

for other types of problems is still required, this was already an important step in proving the feasibility of our lightweight approach.

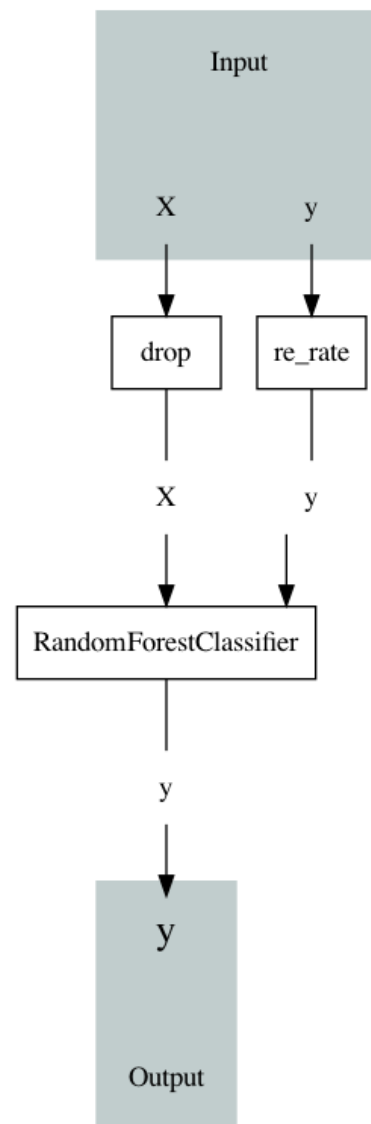


Figure 3.8: Diagram for the steps in our Machine Learning model.

Chapter 4

Experimental Validation

In this Chapter we present the validation process we have submitted our tools to. Starting with Section 4.1, where we provide a brief introduction on the methods we used to evaluate the suitability of our solution, and a comprehensive list of open-source tools which we considered when choosing what alternatives to compare our solution against.

Then, in Section 4.2 we describe our analysis of DAGsHub, a platform for collaboration and development in Machine Learning projects, as well as its direct comparison with our solution. Lastly, in Section 4.3 we showcase the results of applying this same comparison method with Kubeflow, an end-to-end cloud-based Machine Learning Pipeline, and discuss the consequences of its higher infrastructure requirements.

4.1 Comparison Introduction

As we have already demonstrated on the previous chapter, the Lightweight Pipeline is capable of solving standard ML classification problems. However, it is also important to evaluate how our approach compares to the existing solutions currently employed by Machine Learning professionals. Namely, we were interested in measuring the extent to which we were able to decrease the setup and infrastructure complexity, measured in required resources and number of setup steps, as well as how this decrease in infrastructure affects development quality.

For the validation process, we began by gathering the following list of open-source ML platforms and pipelines:

- DAGsHub [64]
- FedML [46]
- H2O [50]
- Hopsworks [36]
- Kubeflow [54]

- LynxKite [58]
- MLReef [38]
- Pachyderm [40]
- Polyaxon [65]

Despite the obvious scale difference between QML and the tools on this list, as many of them have years of development and large teams behind them, we still think it is worthwhile to make this comparisons, as they can help us understand the advantages and disadvantages of our approach.

As it was not possible to experiment on all of the alternative tools, we selected the two most appropriate ones. We chose DAGsHub, in order to provide us with a lightweight baseline for the comparison, and, in contrast, Kubeflow, for a highly complex development environment.

Our analysis began with DAGsHub, where we evaluated its architecture and feature-set, initialized a development environment, and replicated the wine quality project from Section 3.5. Next, we attempted to repeat these steps with Kubeflow, however, we were unable to set-up a proper development environment, as it required expertise and resources outside our reach. As such, the conclusions we are able to make from this process were themselves somewhat limited.

4.2 DAGsHub Analysis

DAGsHub is a software collaboration platform tailored for data scientists and engineers, which enables users to work together on Machine Learning projects, share code and data, and track experiments. It focuses on solving the same problems that tools like GitHub and GitLab solve, however, it is completely adapted to the Machine Learning environment.

DAGsHub is built on top of the Git Version Control system, which versions most of the files in the project, however, much like the Lightweight Pipeline, it also utilises DVC to handle the versioning of data and models, as can be seen in Figure 4.1. Additionally, it also provides plug-ins to seamlessly integrate with MLFlow, which extends DAGsHub's experiment tracking features, and Jenkins, which helps automate CI/CD processes in the pipeline.

Despite DAGsHub itself not being a Machine Learning pipeline, it supports many useful features, and facilitates the development of Machine Learning projects immensely. Due to these reasons, we decided to analyse the development of a Machine Learning project with DAGsHub, in order to provide a lightweight baseline to compare our solution against.

4.2.1 Set-up Process

To start using DAGsHub for any Machine Learning project, first it is necessary to create a DAGsHub account. Akin to GitHub, DAGsHub allows free accounts to use a certain level of basic features, such as unlimited repositories, 10 GB of storage, and access all of the main DAGsHub components.

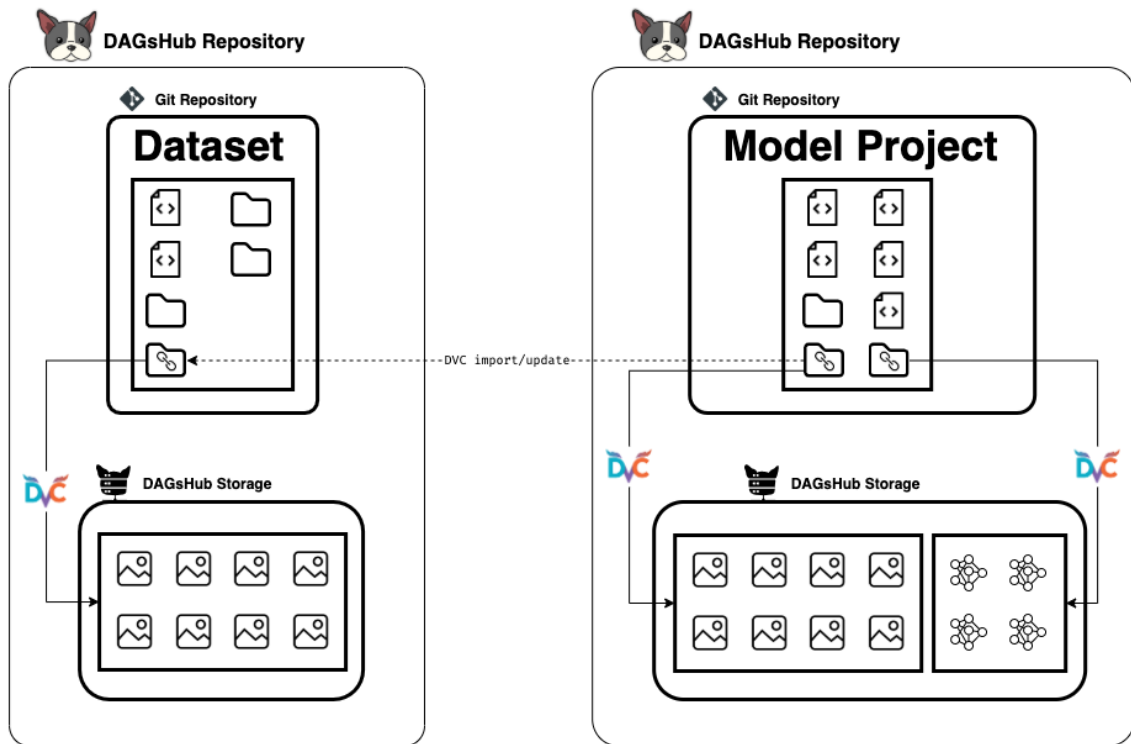


Figure 4.1: Storage Architecture of DAGsHub, presented in [20]

After activating our free account, we created a project repository, which allowed us to choose between two different templates. The chosen template was the "Cookie Cutter DVC Template", as it provided the most comprehensive development environment, and with the repository created, we then proceeded to clone it in our local machine.

After cloning, it was very simple to get started. The project comes with some pre-built scripts that facilitate the setup process, which we ran without problems. Then, we followed the README instructions to complete the remote DVC set-up, and after that, we were ready to start developing with the template's pipeline, which is shown in Figure 4.2.

4.2.2 DAGsHub Experiment

In order to test the DAGsHub workflow, we re-did the Wine Quality project showcased in Section 3.5. To start, we added the "winequality-red.csv" file to our project's "data/raw/" folder, committed the changes to both Git and DVC, and pushed to DAGsHub.

DAGsHub provides a basic tool to explore raw data files, as can be seen in Figure 4.3, however, this tool does not have any graph visualisation features, which were necessary for our needs. To visualise the data, we manually plotted some graphs using the seaborn python package in a Jupyter Notebook, which gave us some Figures like 4.4 and 4.5.

After completing the data analysis, we processed the data-set by modifying the target variable's scale to 0 - 2, and dropping the "pH" and "free sulfur dioxide" columns, just as we had done in the

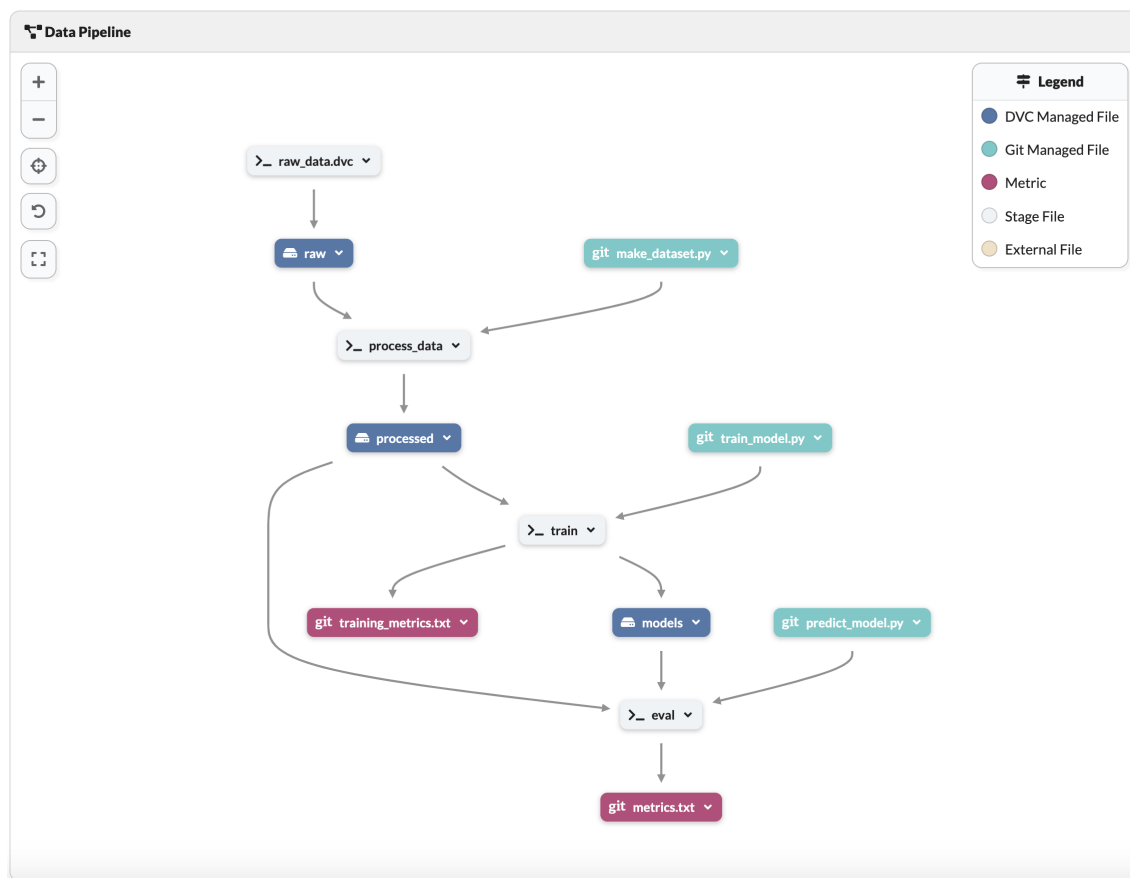


Figure 4.2: Pre-built pipeline generated by DAGsHub "Cookie Cutter DVC Template", in our repository.

winequality-red-FE.csv 83 KB								Permalink	Raw	
#		fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur d...			
1	0	7.4	0.7	0.0	1.9	0.076	34.0			
2	1	7.8	0.88	0.0	2.6	0.098	67.0			
3	2	7.8	0.76	0.04	2.3	0.092	54.0			
4	3	11.2	0.28	0.56	1.9	0.075	60.0			
5	4	7.4	0.7	0.0	1.9	0.076	34.0			
6	5	7.4	0.66	0.0	1.8	0.075	40.0			
7	6	7.9	0.6	0.06	1.6	0.069	59.0			

Figure 4.3: Data exploration tool provided by DAGsHub, in our repository.

Lightweight Pipeline demonstrations. Since DAGsHub does not provide a primitive system like MLBazaar, we simply coded these modifications in a python file, and then executed them with the

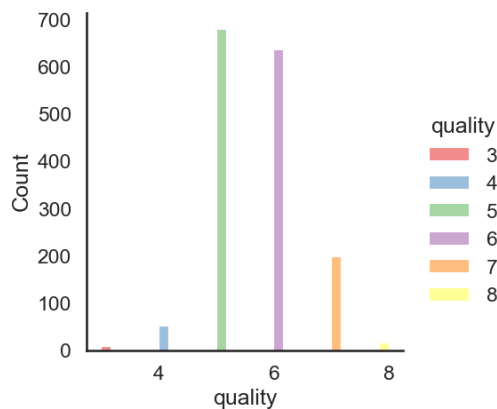


Figure 4.4: Plot of class distribution in wine quality data-set.

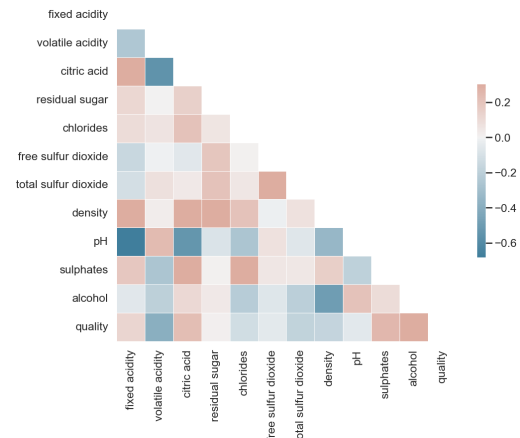


Figure 4.5: Plot of correlation Matrix in wine quality data-set.

command "dvc repro process_data", which automatically versioned the outputs.

After analysing the resultant data to check whether the changes had been successful, we committed them, and created a new branch to perform the first model experiment.

In the first experiment branch we tried a Logistic Regression model, for which we had to code its training session and test session. To execute this experiment we ran the pipelines "train" and "eval" steps using DVC's CLI, and pushed the results to DAGsHub, where they were automatically tracked. We then repeated these steps in a new branch for each of our experiments, and analysed the results in DAGsHub's Web Interface, as shown in Figure 4.6. Finally, due to the lack of support for the deployment phase in DAGsHub, we concluded the development of the project, and constructed a workflow diagram from our experiment, displayed in Figure 4.7.

Compare Reset filters Archive Labels Columns											
☐	Code	Name	Labels ▾	Commit	Cre... ↓	Sou...	Train Time	Accuracy	MSE	F1	RMSE
☐		flower swallow	random forest	25e4	15 hours ago		0.0588510036...	0.865	0.135	0.8514977933...	0.3674234614...
☐		feather koala	logistic regression	4f35	15 hours ago		0.0165848731...	0.835	0.165	0.7842935606...	0.4062019202...
☐		field monkey	naive bayes	3577	15 hours ago		0.0025198459...	0.805	0.195	0.8128650428...	0.4415880433...
☐		smoke ferret	xgbclassifier	a27d	15 hours ago		0.1958880424...	0.8375	0.1625	0.8083067047...	0.4031128874...

Figure 4.6: Experiment Tracking of our Wine Quality Project in DAGsHub Web UI.

4.2.3 QML vs. DAGsHub

From this experiment, we were able to analyse how our QML + Lightweight Pipeline tool-set compares against a regular DAGsHub environment. Starting with the Setup process, DAGsHub's setup is slightly longer than ours, as can be seen by comparing the Figures 3.3 and 4.7. Despite this small difference, we find both systems equally as acceptable, since most of the setup steps are semi-automated, and simple enough to debug in case something goes wrong.

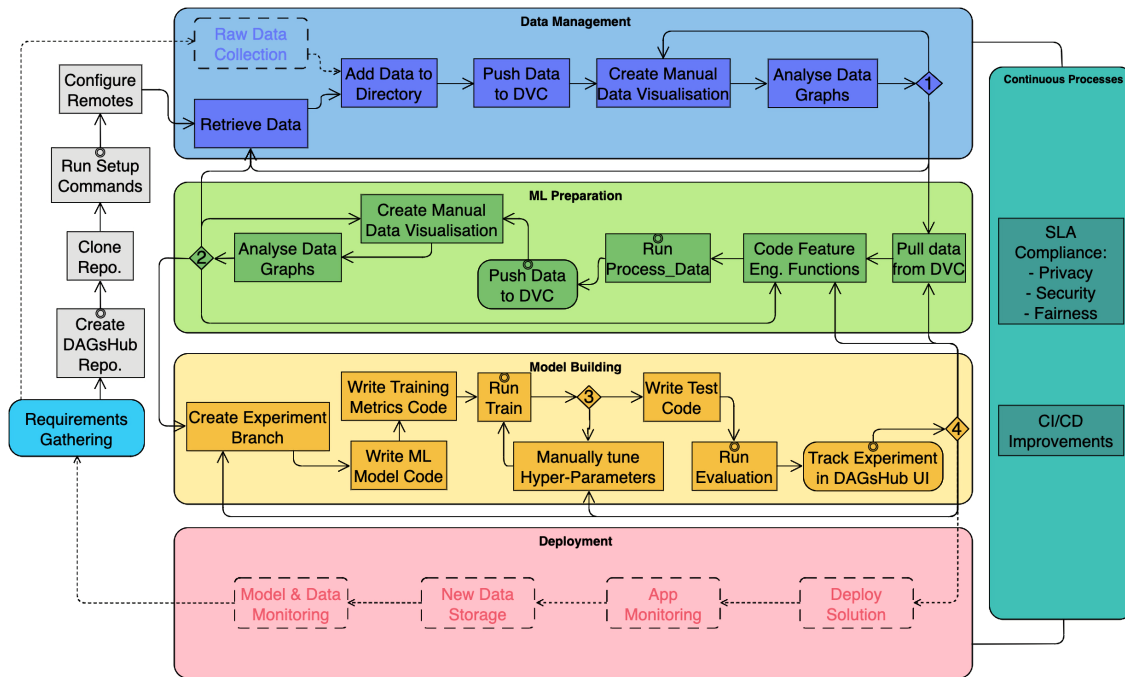


Figure 4.7: Workflow Diagram proposed for DAGsHub project development.

With the experiment complete, we were able to compare the development process experienced in DAGsHub with the development process of QML, by comparing the two flowcharts in Figures 3.3 and 4.7.

As shown in Table 4.1 the setup and required infrastructure for both approaches is very similar. Both only require a local computer to operate, and even though DAGsHub’s workflow requires 2 extra steps for the setup, these are all very simple to execute, as such, we consider both tools equivalent in terms of overhead.

	QML’s Lightweight Pipeline	DAGsHub’s Cookie Cutter DVC Template
Required infrastructure	Local Machine	Local Machine
N° setup steps	2	4
Setup overhead	Low	Low

Table 4.1: Setup and infrastructure comparison between QML and DAGsHub

On the ML development itself, DAGsHub provides the user with a much simpler pipeline that only provides support for the first three development phases, with 4 decision nodes and 19 steps, 2 of which are fully automated, and 3 are semi-automated. In contrast, the Lightweight Pipeline supports all development phases and has a total of 7 decision nodes and 29 steps, 7 of which are fully automated, and 8 semi-automated. Due to this simplistic approach, the number of supported

features by DAGsHub's pipeline is smaller than in the Lightweight Pipeline, as can be seen in Table 4.2, where we compare the support for each of the 11 pipeline features presented at the end of chapter 2.

	QML's Lightweight Pipeline	DAGsHub's Cookie Cutter DVC Template
Data Versioning	Automated	Assisted
Data Analysis	Semi-Automated	No Support
Data Cleaning	Assisted	Assisted
Feature Engineering	Assisted	Assisted
Model Training	Assisted	Assisted
Hyper-Parameter Tuning	Semi-Automated	No Support
Model Evaluation	Semi-Automated	Assisted
Model Versioning	Automated	Automated
Experiment Tracking	No Support	Automated
Model Deployment	Semi-Automated	No Support
Model Monitoring	No Support	No Support

Table 4.2: Feature and automation comparison between QML and DAGsHub. "Not Supported" for tasks that are not at all supported by the pipeline. "Assisted" for tasks that are mostly manual, but have some pipeline support. "Semi-Automated" for tasks that are mostly automated but require some sort of user input or direction. "Automated" for tasks that are completely automated and require no user direction.

As the comparison table shows, the Lightweight Pipeline supports 9 out of 11 features, against DAGsHub's 7. Additionally, the level of automation on the Lightweight Pipeline is higher, as it fully automates two tasks and semi-automates 4, compared to only 2 fully automated tasks by DAGsHub's pipeline.

Lastly, it is important to check whether the different pipelines altered the prediction results in any way. In Table 4.3 we have grouped together the scores of the equivalent models on both pipelines. Generally the scores are aligned, even if with a very slight deviation in favour of the Lightweight Pipeline, but we consider this to be too marginal to be significant, and so we can say that both pipelines are equivalent in model performance.

Aside from these quantifiable results, it is also worth pointing out some qualitative differences between the two pipelines. First, despite both of them assisting with Data Cleaning, Feature Engineering and Model Training, our approach produces much less glue code, as we base the development of the model pipelines on MLBazaar [60]. Furthermore, DAGsHub's branch-based approach for experiments results in a lot of duplicated code between branches, which proves detrimental to development when it is necessary to change something across branches, such as an evaluation metric.

On the up-side however, DAGsHub takes care of the remote repository hosting, and contains an helpful web UI, which make it easier to manage and visualise the pipeline.

		QML's Lightweight Pipeline	DAGsHub's Cookie Cutter DVC Template
Random Forest	Accuracy	0.88	0.87
	MSE	0.12	0.14
	RMSE	0.35	0.37
	F1	0.87	0.85
Logistic Regression	Accuracy	0.85	0.84
	MSE	0.15	0.17
	RMSE	0.39	0.41
	F1	0.78	0.78
Naive Bayes Classifier	Accuracy	0.81	0.81
	MSE	0.19	0.20
	RMSE	0.44	0.44
	F1	0.76	0.81
XGBoost Classifier	Accuracy	0.85	0.84
	MSE	0.15	0.16
	RMSE	0.39	0.40
	F1	0.83	0.81

Table 4.3: ML Model results comparison between the Lightweight Pipeline and DAGsHub's pipeline.

Finally, both options are similarly extensible, as it is possible to edit DAGsHub's DVC files to change the pipelines functionalities somewhat. The main difference being that our approach allows the developers to automatically monitor directory events, and add new CLI commands to enhance human interaction.

In conclusion, we found that the Lightweight Pipeline provides a more automated and complete support for the ML development process. However, DAGsHub distinguishes itself by providing hosting for the remote repositories, and nice web UIs which help with collaboration and Experiment Tracking.

Additionally, one interesting finding that this analysis revealed, is that it would not require many changes to the Lightweight Pipeline to integrate it with DAGsHub's hosting and Experiment Tracking services, which might be an interesting area for future work.

4.3 Kubeflow Analysis

Kubeflow is an open-source Machine Learning Toolkit, which integrates several Machine Learning frameworks for each of the development phases, in a cohesive Kubernetes environment. Its architecture is based on microservices, which allows for a loosely coupled system where each component can be independently updated and scaled, and then uniformly orchestrated with Kubernetes.

The main components that support Kubeflows operations are: [43]

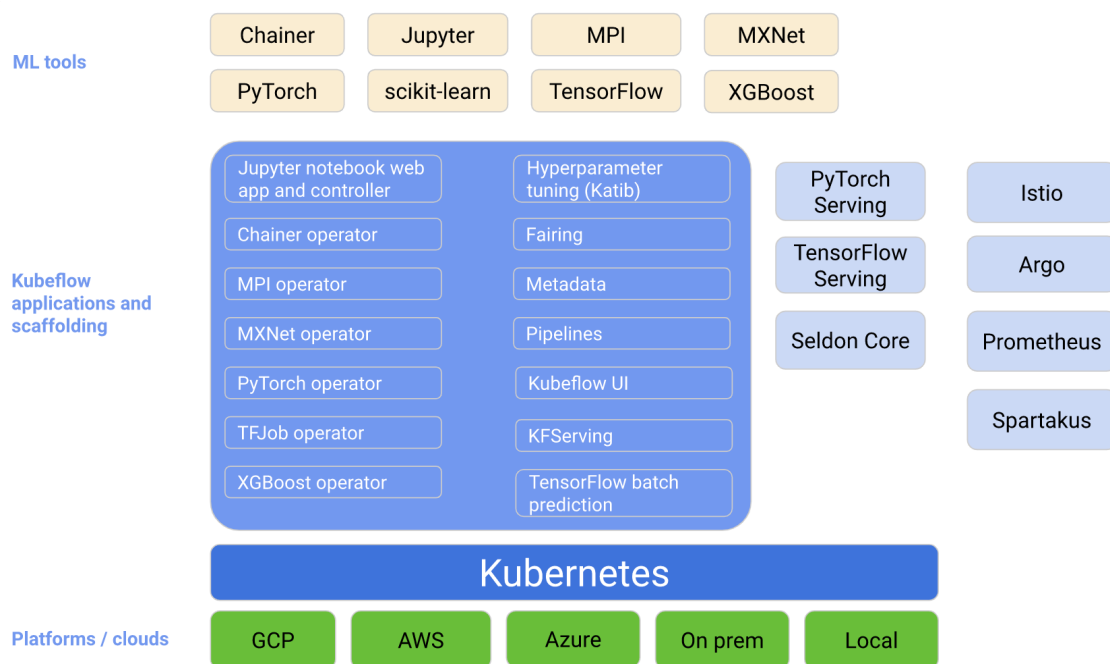


Figure 4.8: KubeFlows component Architecture, as presented in [43]

- **Argo:** The workflow engine that allows users to define and run complex multi-step processes on Kubernetes.
- **Istio:** The platform that connects, manages, and secures Kubernetes microservices.
- **Knative:** An open-source project that provides a set of serverless workload primitives, and tools to build, deploy, and run those workloads on Kubernetes.
- **Katib:** A hyperparameter tuning framework for KubeFlow.
- **Seldon Core:** An open-source project that provides a set of tools to build, deploy, and manage Machine Learning models on Kubernetes.
- **JupyterHub:** A shared multi-user Jupyter Notebook environment.

Aside from these components, KubeFlow also supports an extensive collection of Machine Learning libraries, as can be seen in figure 4.8.

At its core, KubeFlow is a platform built for cloud environments, which already poses a significant barrier to entry for new users, as it can be expensive to acquire and maintain such an infrastructure. However, as can be seen in Figure 4.8, it also supports local installations, and since QML was designed for local environments, we chose this approach for KubeFlow as well. Unfortunately, even when following KubeFlow’s own setup guide [56], it was not at all straightforward to get the service up and running in a local environment.

First, it was necessary to create an Ubuntu virtual environment using Multipass [63], and then, inside this virtual environment we deployed Kubernetes with Microk8s[61], where we tried to deploy Kubeflow. However, even after many hours spent trying to get this approach to work, we were unsuccessful.

As some of the problems encountered seemed to be due to a lack of resources in the local machine, we attempted to install kubeflow-lite [45]. But even with this variant, the setup got stuck in the boot-up process for several hours, and we chose to abandon the approach.

On our most successful attempt, we utilised kind [53] to host the Kubernetes environment, and instead of deploying the entire Kubeflow platform, we only deployed its Pipeline module. However, our local machine still did not meet the minimum required resources of 40GB of Memory, 4 CPU Cores and 12GB of RAM, which prevented us from successfully running any experiments on the deployed pipelines.

On our final attempt, we tried to deploy Kubeflow on a dedicated private server, however, not even this approach proved successful, as it kept on running into Kubernetes-related problems, one after the other.

During our search for a solution, we confirmed that this experience with Kubeflow was not unique. Various other sources reported that it was not a trivial matter to operate a Kubeflow instance, and that this platform should only be employed with the support of a dedicated team experienced in Kubernetes [55], [25]. In fact, this is such a glaring issue, that it even gave rise to a Kubeflow as a service products [35].

Since we did not have access to the resources or expertise required to operate Kubeflow, we concluded that it was not worthwhile to continue this experiment further, and, as corroborated by other sources, despite Kubeflow's interesting features, it has too much infrastructure complexity to be useful for small teams, in the initial stages of ML development.

Chapter 5

Discussion

In this chapter we discuss our accomplishments in this dissertation, and assess the extent to which we were able to answer our initial research questions in Section 5.1. Then, in Section 5.2, we explore some of the limitations of our approach, and present some possible future research directions based on the work we have conducted. An lastly, we close this dissertation with a final note, on Section 5.3.

5.1 Conclusions

We started this dissertation with the main objective of developing an end-to-end ML pipeline, with the added requirements of high extensibility and low setup / maintenance costs. This culminated in three research questions:

- **RQ1:** What are the required functionalities of a Machine Learning pipeline?
- **RQ2:** How feasible is it to create modular pipelines that automate time-consuming Machine Learning tasks, without significantly increasing infrastructure complexity?
- **RQ3:** How does a lightweight pipeline perform in comparison to existing alternatives?

We answered RQ1 in chapter 2, with our research into the background and state of the art on the fields of MLOPs and AutoML, which culminated in a 11 item list of required pipeline features, as well as 4 recommendations for pipeline characteristics.

In chapter 3, we tackled RQ2 by developing a framework for ML pipeline creation (QML), and a simple Lightweight Pipeline, which supported 9 out of the 11 features previously mentioned. In order to validate our approach, we demonstrated the development process for a standard ML classification problem using our pipeline, thus confirming that it is feasible to create effective and modular low-infrastructure pipelines, for Machine Learning development.

Lastly, to answer our third research question (RQ3), an experimental comparison between our tools and two other existing alternatives was conducted in chapter 4. From this analysis we concluded that, when compared to the lightweight alternative (DAGsHub), our pipeline presented

greater automation and feature support, at roughly the same level of setup / maintenance overhead. When compared to Kubeflow, we were only able to conclude that it had the most setup and maintenance costs, many times larger than our own, which prevented us from continuing its experiment.

5.2 Future Work

Despite the successes we have found in this project, there are some limitations and drawbacks from our current approach which we will discuss in the following subsections, as well as possible future research directions, which we deem most appropriate.

5.2.1 Future Work on the QML Framework

QML was proposed with the goal of simplifying the pipeline creation process. We have accomplished this goal by making it possible to extend the capabilities of any given pipeline through simple edits to its YAML description file. However, despite following this method, in practice it is usually also necessary to code the required functionality with python. And while this necessity will never be fully averted, it can be minimized by making it simpler to share and re-utilize components created in other pipelines.

Currently, our approach restricts the pipeline developer to the components present inside his project. In which case, if a component from a different project is desired, it has to be manually copied into the new pipeline, which can result in duplicate code files on the system.

Additionally, while our pipeline encourages modular development, there are some important edge-cases which it leaves unaddressed. For example, in our Lightweight pipeline we have created a module called "add_to_DVC", which automatically versions data files with DVC. However, If someone were to re-utilize this module in another pipeline, he would most likely have to change some of the source code. This would be necessary because our module utilizes hard-coded values to save the versioning files on a folder, which in our case is entitled `"/data_conf/"`. If the new hypothetical pipeline wanted to utilize a different folder name, it would need to alter the source-code, thus creating a new component, instead of re-utilizing an existing one. Furthermore, to utilize this module it is first necessary to initialize DVC, which is done inside our "project_initializer" component. As such a requirement is not specified anywhere by QML, it could lead to several frustrations when integrating modules from other pipelines.

These limitations point us to the first possible future work direction. Improving pipeline modularity and re-usability by creating of a Hub for pipelines and their modules, which should be accessible through the CLI, and take the place of the "edit" command.

The second line of possible future research, paradoxically passes through increasing infrastructure support, the part of ML pipeline which we have tried to minimize in this project.

Through our efforts to make ML pipelines more accessible, we have ignored the use-cases that actually need heavy infrastructure support, such as external servers, or cloud environments. However, while the Lightweight Pipeline was effective without these resources, we also recognize that

as projects grow, so must their pipelines. Which means that a fully local solution will eventually reach the end of its usefulness, as the ML teams grow, or models become larger and harder to train on local machines.

As it stands, if a QML pipeline ever required to utilize cloud resources or an external server, that part of the pipeline would have to be developed outside QML, even if the communication with said resources could happen through QML commands. With this in mind, in order to ensure that pipeline complexity remains as low as possible, future research will have to focus on supporting the development of remote pipelines with the same kind of reproducibility and ease that we currently provide for local ones.

5.2.2 Future Work on the Lightweight Pipeline

After the demonstrations we have performed in this dissertation, it became apparent that there were some aspects of our Lightweight Pipeline that could be improved upon. First, in order to provide support for all the features identified in chapter 2, it is necessary to add Experiment Tracking, and Model Monitoring capabilities. The first of which could be easily done by shifting our development environment to work in tandem with DAGsHub, as it proved to be an excellent lightweight tool. This would restrict developers somewhat, on their remote repository choice, however, it would also add remote storage features that we would otherwise be unable to provide, which is a sufficient upside to warrant the restriction.

Another aspect that could be improved, is our technology choice. In order to eliminate glue code and simplify the model experiments on the ML Preparation and Model Building phases, we relied heavily upon MLBazaar [17]. MLBazaar is able to join together ML primitives from many different libraries by assigning a universal JSON annotation schema to each one, which is a very interesting and useful proposition. However, our problem lies with its current development state. Despite finding great success at solving the glue code problem, this package is currently in a Pre-Alpha stage, and lacks many quality of life features, as well as support for crucial ML primitives that we find necessary. Both of these factors combined can sometimes stall or restrict development, as it might be necessary to port a simple primitive into MLBazaar's intricate annotation format, or even go in a completely separate direction altogether, due to some other the limitation with the library. As such, despite showing great promise, it might be necessary to look for other alternatives while MLBazaar's development does not pick up.

Lastly, the main short-coming of this dissertation was the comparison process between the Lightweight Pipeline and the different alternative tools, as we were only able to complete the comparison experiment with DAGsHub. As such, we think it would be interesting to conduct a larger-scale experimental study on current ML platforms and pipelines, with the inclusion of QML, in order to understand the advantages and disadvantages of each approach, and when to use them.

5.3 Closing Note

Throughout the course of this dissertation, the presented tools went through a series of iterations which have culminated in the current form they take today. While there is certainly still a lot of room for further growth, we believe that we have met our goals, and successfully demonstrated how QML and the Lightweight Pipeline can enhance the development experience of smaller ML projects, without burdening the teams with unnecessarily complex infrastructure.

Appendix A

Lightweight Pipeline Screenshots

In this chapter we have included some screenshots of our pipeline in action, which would difficult the reading process if placed amidst the rest of the dissertation.

A.1 Inspecting Data

A.1.1 Automatically Generated Report

A.1.2 Deployed Model Test

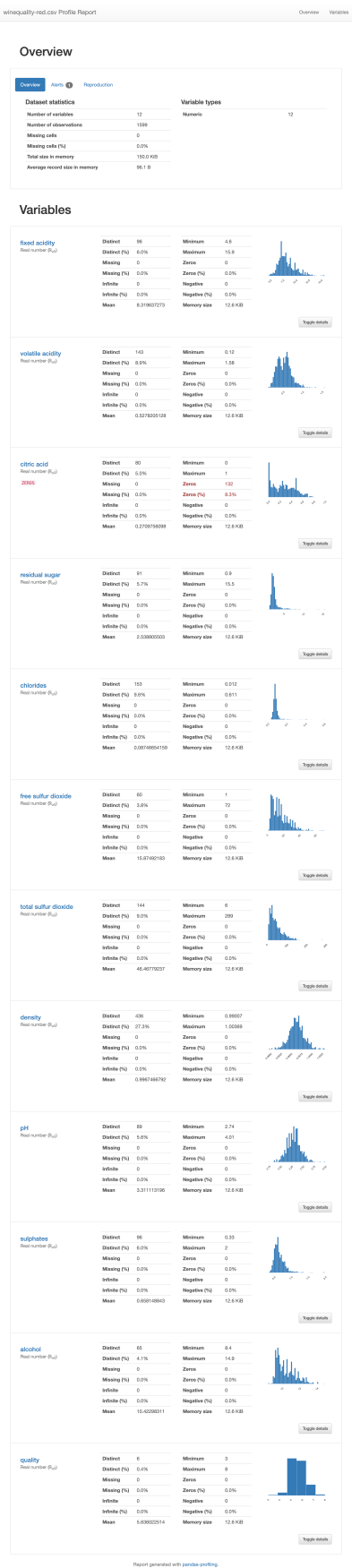


Figure A.1: Example of a data report automatically generated when a data file is added or modified.

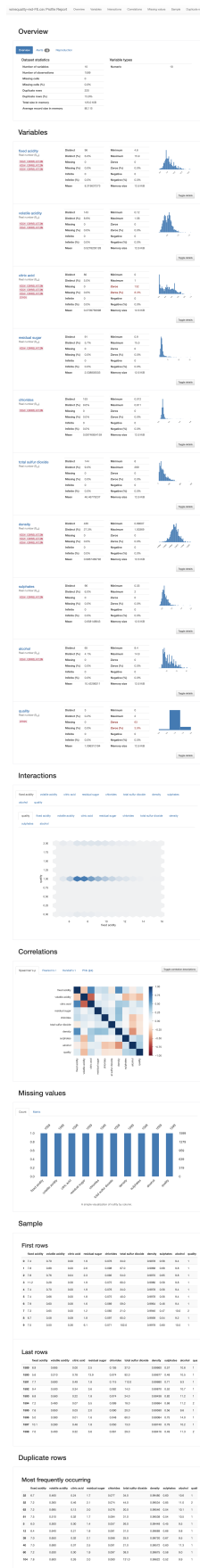


Figure A.2: Example of a full report generated with our inspect_data command.

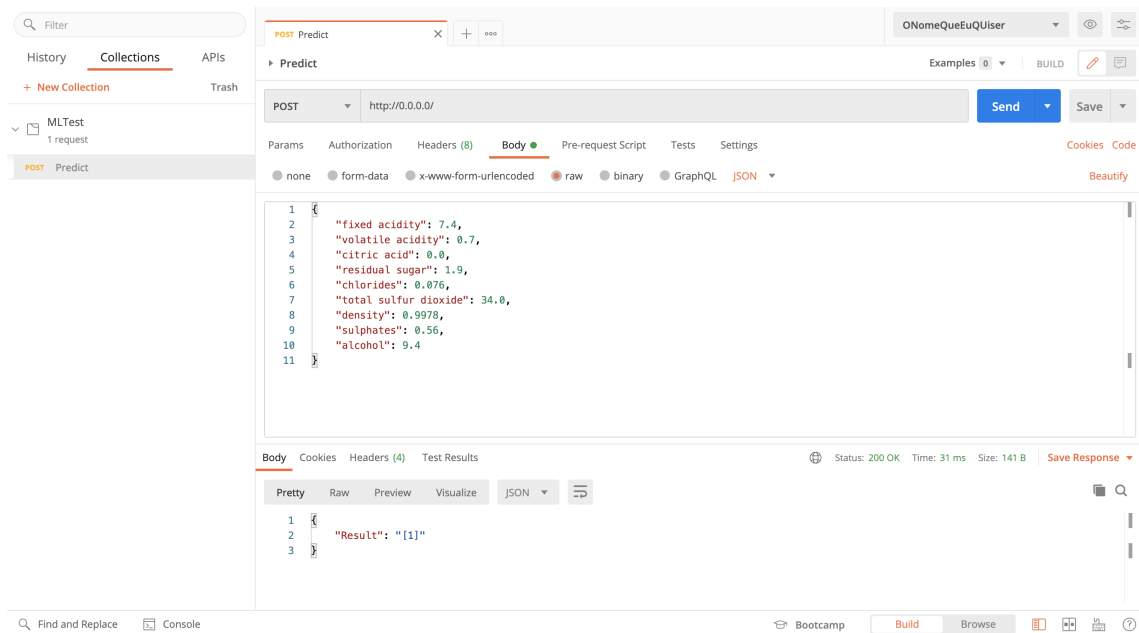


Figure A.3: Example of a live prediction made after the deployment of our Wine quality model, built with the Lightweight Pipeline.

References

- [1] Paulo Cortez et al. “Modeling Wine Preferences by Data Mining from Physicochemical Properties”. In: *Decision Support Systems*. Smart Business Networks: Concepts and Empirical Evidence 47.4 (Nov. 1, 2009), pp. 547–553. ISSN: 0167-9236. DOI: [10.1016/j.dss.2009.05.016](https://doi.org/10.1016/j.dss.2009.05.016). URL: <https://www.sciencedirect.com/science/article/pii/S0167923609001377> (visited on 06/22/2022).
- [2] D. Sculley et al. “Hidden Technical Debt in Machine Learning Systems”. In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015. URL: <https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fc2674f757a246Abstract.html> (visited on 02/21/2022).
- [3] Gene Kim et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. First edition. Portland, OR: IT Revolution Press, LLC, 2016. 437 pp. ISBN: 978-1-942788-00-3.
- [4] Joakim Verona. *Practical DevOps*. Packt Publishing Ltd, Feb. 16, 2016. 240 pp. ISBN: 978-1-78588-652-2. Google Books: [aK1KDAAAQBAJ](https://books.google.com/books?id=aK1KDAAAQBAJ).
- [5] Pranoy Radhakrishnan. *What Are Hyperparameters ? And How to Tune the Hyperparameters in a Deep Neural Network?* Medium. Oct. 18, 2017. URL: <https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a> (visited on 06/05/2022).
- [6] Dominik Sacha et al. “What You See Is What You Can Change: Human-centered Machine Learning by Interactive Visualization”. In: *Neurocomputing*. Advances in Artificial Neural Networks, Machine Learning and Computational Intelligence 268 (Dec. 13, 2017), pp. 164–175. ISSN: 0925-2312. DOI: [10.1016/j.neucom.2017.01.105](https://doi.org/10.1016/j.neucom.2017.01.105). URL: <https://www.sciencedirect.com/science/article/pii/S0925231217307609> (visited on 06/09/2022).
- [7] Ahmad EL Sallab et al. “Deep Reinforcement Learning Framework for Autonomous Driving”. In: *Electronic Imaging 2017.19* (Jan. 29, 2017), pp. 70–76. DOI: [10.2352/ISSN.2470-1173.2017.19.AVM-023](https://doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023).

- [8] Pratibha Jha and Rizwan Khan. “A Review Paper on DevOps: Beginning and More To Know”. In: *International Journal of Computer Applications* 180.48 (June 15, 2018), pp. 16–20. ISSN: 09758887. DOI: [10.5120/ijca2018917253](https://doi.org/10.5120/ijca2018917253). URL: <http://www.ijcaonline.org/archives/volume180/number48/jha-2018-ijca-917253.pdf> (visited on 02/21/2022).
- [9] Will Koehrsen. *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. Medium. July 2, 2018. URL: <https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f> (visited on 06/05/2022).
- [10] Agata Nawrocka, Andrzej Kot, and Marcin Nawrocki. “Application of Machine Learning in Recommendation Systems”. In: *2018 19th International Carpathian Control Conference (ICCC)*. 2018 19th International Carpathian Control Conference (ICCC). May 2018, pp. 328–331. DOI: [10.1109/CarpathianCC.2018.8399650](https://doi.org/10.1109/CarpathianCC.2018.8399650).
- [11] *Add It Up: How Long Does a Machine Learning Deployment Take?* The New Stack. Dec. 19, 2019. URL: <https://thenewstack.io/add-it-up-how-long-does-a-machine-learning-deployment-take/> (visited on 02/28/2022).
- [12] Tom Diethe et al. “Continual Learning in Practice”. Mar. 18, 2019. DOI: [10.48550/arXiv.1903.05202](https://doi.org/10.48550/arXiv.1903.05202). arXiv: 1903.05202 [cs, stat]. URL: <http://arxiv.org/abs/1903.05202> (visited on 06/09/2022).
- [13] *IDC: For 1 in 4 Companies, Half of All AI Projects Fail*. VentureBeat. July 8, 2019. URL: <https://venturebeat.com/2019/07/08/idc-for-1-in-4-companies-half-of-all-ai-projects-fail/> (visited on 02/28/2022).
- [14] ODSC-Open Data Science. *The Past, Present, and Future of Automated Machine Learning*. Medium. Aug. 15, 2019. URL: <https://odsc.medium.com/the-past-present-and-future-of-automated-machine-learning-5e081ca4b71a> (visited on 03/01/2022).
- [15] Radwa El Shawi, Mohamed Maher, and S. Sakr. “Automated Machine Learning: State-of-The-Art and Open Challenges”. In: *undefined* (2019). URL: <https://www.semanticscholar.org/paper/Automated-Machine-Learning%3A-State-of-The-Art-and-Shawi-Maher/663108c231afdb91ca1e8af8ef8a6a937b5a6e20> (visited on 12/14/2021).
- [16] Karansingh Chauhan et al. “Automated Machine Learning: The New Wave of Machine Learning”. In: *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA). Bangalore, India: IEEE, Mar. 2020, pp. 205–212. ISBN: 978-1-72814-167-1. DOI: [10.1109/ICIMIA48430.2020.9074859](https://doi.org/10.1109/ICIMIA48430.2020.9074859). URL: <https://ieeexplore.ieee.org/document/9074859/> (visited on 03/01/2022).

- [17] Micah J. Smith et al. “The Machine Learning Bazaar: Harnessing the ML Ecosystem for Effective System Development”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’20. New York, NY, USA: Association for Computing Machinery, June 11, 2020, pp. 785–800. ISBN: 978-1-4503-6735-6. DOI: [10.1145/3318464.3386146](https://doi.org/10.1145/3318464.3386146). URL: <https://doi.org/10.1145/3318464.3386146> (visited on 02/22/2022).
- [18] Mark Treveil et al. *Introducing MLOps*. "O'Reilly Media, Inc.", Nov. 30, 2020. 171 pp. ISBN: 978-1-09-811642-2. Google Books: [CCoMEAAQBAJ](#).
- [19] *Data-Centric Approach vs Model-Centric Approach in Machine Learning*. neptune.ai. Dec. 30, 2021. URL: <https://neptune.ai/blog/data-centric-vs-model-centric-machine-learning> (visited on 06/07/2022).
- [20] *Datasets Should Behave like Git Repositories*. DagsHub Blog. Jan. 18, 2021. URL: <https://dagshub.com/blog/datasets-should-behave-like-git-repositories/> (visited on 06/26/2022).
- [21] DeepLearningAI, director. *A Chat with Andrew on MLOps: From Model-centric to Data-centric AI*. Mar. 24, 2021. URL: <https://www.youtube.com/watch?v=06-AZXmwHjo> (visited on 06/07/2022).
- [22] Dejan Golubovic and Ricardo Rocha. “Training and Serving ML Workloads with Kubeflow at CERN”. In: *EPJ Web of Conferences* 251 (2021), p. 02067. ISSN: 2100-014X. DOI: [10.1051/epjconf/202125102067](https://doi.org/10.1051/epjconf/202125102067). URL: https://www.epj-conferences.org/articles/epjconf/abs/2021/05/epjconf_chep2021_02067/epjconf_chep2021_02067.html (visited on 06/03/2022).
- [23] Idil Ismiguzel. *Hyperparameter Tuning with Grid Search and Random Search*. Medium. Sept. 29, 2021. URL: <https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144> (visited on 06/05/2022).
- [24] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. “Towards MLOps: A Framework and Maturity Model”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Sept. 2021, pp. 1–8. DOI: [10.1109/SEAA53835.2021.00050](https://doi.org/10.1109/SEAA53835.2021.00050).
- [25] Mateusz Kwaśniak. *Kubeflow (Is Not) for Dummies*. Medium. June 17, 2021. URL: <https://towardsdatascience.com/kubeflow-is-not-for-dummies-414d8977158a> (visited on 07/04/2022).
- [26] Sasu Mäkinen et al. “Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?” Mar. 16, 2021. arXiv: [2103.08942](https://arxiv.org/abs/2103.08942) [cs]. URL: <http://arxiv.org/abs/2103.08942> (visited on 01/07/2022).

- [27] Varón Maya and Andrés Felipe. “The State of MLOps”. In: (2021). URL: <https://repositorio.uniandes.edu.co/handle/1992/51495> (visited on 02/17/2022).
- [28] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. “Challenges in Deploying Machine Learning: A Survey of Case Studies”. Jan. 18, 2021. arXiv: 2011.09926 [cs]. URL: <http://arxiv.org/abs/2011.09926> (visited on 02/17/2022).
- [29] EMMANUEL RAJ. *MLOPS USING AZURE MACHINE LEARNING: Rapidly Test, Build, and Manage Production-Ready Machine... Learning Life Cycles at Scale*. S.I.: PACKT PUBLISHING LIMITED, 2021. ISBN: 978-1-80056-288-2.
- [30] Philipp Ruf et al. “Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools”. In: *Applied Sciences* 11.19 (19 Jan. 2021), p. 8861. DOI: 10.3390/app11198861. URL: <https://www.mdpi.com/2076-3417/11/19/8861> (visited on 01/07/2022).
- [31] Yiming Tang et al. “An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). May 2021, pp. 238–250. DOI: 10.1109/ICSE43902.2021.00033.
- [32] Dakuo Wang et al. “How Much Automation Does a Data Scientist Want?” Jan. 6, 2021. arXiv: 2101.03970 [cs]. URL: <http://arxiv.org/abs/2101.03970> (visited on 02/17/2022).
- [33] Marc-André Zöller and Marco F. Huber. “Benchmark and Survey of Automated Machine Learning Frameworks”. In: *Journal of Artificial Intelligence Research* 70 (Jan. 27, 2021), pp. 409–472. ISSN: 1076-9757. DOI: 10.1613/jair.1.11854. URL: <https://www.jair.org/index.php/jair/article/view/11854> (visited on 02/16/2022).
- [34] *Automate and Scale Your Machine Learning | Pachyderm*. Feb. 21, 2022. URL: <https://www.pachyderm.com/> (visited on 06/04/2022).
- [35] *Introducing Kubeflow as a Service Giving Data Scientists Instant Access to a Complete MLOps Platform*. Accelerate Models to Market with Arrikto. May 10, 2022. URL: <https://www.arrikto.com/blog/introducing-kubeflow-as-a-service-giving-data-scientists-instant-access-to-a-complete-mlops-platform/> (visited on 07/03/2022).
- [36] *Logicalclocks/Hopsworks*. Hopsworks, June 26, 2022. URL: <https://github.com/logicalclocks/hopsworks> (visited on 06/26/2022).
- [37] Vinh Luu. *What Is DevOps Lifecycle? Benefits and Method - Bestarion*. Apr. 7, 2022. URL: <https://bestarion.com/devops-lifecycle/>, %20https://bestarion.com/devops-lifecycle/ (visited on 06/02/2022).
- [38] *MLReef/MLreef*. MLReef, June 24, 2022. URL: <https://github.com/MLReef/mlreef> (visited on 06/26/2022).

- [39] Nikolay O. Nikitin et al. “Automated Evolutionary Approach for the Design of Composite Machine Learning Pipelines”. In: *Future Generation Computer Systems* 127 (Feb. 2022), pp. 109–125. ISSN: 0167739X. DOI: 10.1016/j.future.2021.08.022. arXiv: 2106.15397. URL: <http://arxiv.org/abs/2106.15397> (visited on 02/15/2022).
- [40] *Pachyderm – The Leader in Data Versioning and Pipelines for MLOps*. Pachyderm, June 26, 2022. URL: <https://github.com/pachyderm/pachyderm> (visited on 06/26/2022).
- [41] Kelvin S. do Prado. *Awesome MLOps*. Mar. 14, 2022. URL: <https://github.com/kelvins/awesome-mlops> (visited on 03/15/2022).
- [42] *API Reference — Watchdog 0.8.2 Documentation*. URL: <https://pythonhosted.org/watchdog/api.html> (visited on 06/11/2022).
- [43] *Architecture*. Kubeflow. URL: <https://www.kubeflow.org/docs/started/architecture/> (visited on 06/24/2022).
- [44] *Contributing Custom Primitives — MLPrimitives 0.3.2 Documentation*. URL: <https://mlbazaar.github.io/MLPrimitives/community/custom.html> (visited on 06/20/2022).
- [45] *Deploy Kubeflow Lite Using Charmhub - The Open Operator Collection*. URL: <https://charmhub.io/kubeflow-lite> (visited on 06/26/2022).
- [46] *FedML Inc.* GitHub. URL: <https://github.com/FedML-AI> (visited on 06/26/2022).
- [47] Matthias Feurer et al. “Efficient and Robust Automated Machine Learning”. In: (), p. 9.
- [48] *Full Stack Machine Learning Operating System | Cnvr.io*. URL: <https://cnvrg.io/> (visited on 06/04/2022).
- [49] “Global AI Survey: AI Proves Its Worth, but Few Scale Impact”. In: (), p. 11.
- [50] *H2O.AI*. GitHub. URL: <https://github.com/h2oai> (visited on 06/26/2022).
- [51] Stanford University HAI. *AI Index Report*. Tableau Software. URL: https://public.tableau.com/views/ResearchDevelopment_16145904716170/1_1_1b?:embed=y&:showVizHome=no&:host_url=https%3A%2F%2Fpublic.tableau.com%2F&:embed_code_version=3&:tabs=no&:toolbar=no&:animate_transition=yes&:display_static_image=no&:display_spinner=no&:display_overlay=yes&:display_count=yes&:language=en&:loadOrderID=0 (visited on 02/28/2022).
- [52] *Home - Knative*. URL: <https://knative.dev/docs/> (visited on 06/04/2022).
- [53] *Kind*. URL: <https://kind.sigs.k8s.io/> (visited on 06/26/2022).
- [54] *Kubeflow*. Kubeflow. URL: <https://www.kubeflow.org/> (visited on 06/04/2022).
- [55] *Kubeflow: Not Ready for Production?* URL: <https://www.datarevenue.com/en-blog/kubeflow-not-ready-for-production> (visited on 07/03/2022).

- [56] *Local Deployment*. Kubeflow. URL: <https://www.kubeflow.org/docs/components/pipelines/installation/localcluster-deployment/> (visited on 06/25/2022).
- [57] Mike Loukides. “AI Adoption in the Enterprise 2021”. In: (), p. 26.
- [58] *LynxKite*. URL: <https://lynxkite.com/> (visited on 06/26/2022).
- [59] *Machine Learning – Amazon Web Services*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/sagemaker/> (visited on 06/04/2022).
- [60] *Machine Learning Bazaar*. URL: <https://mlbazaar.github.io/> (visited on 06/08/2022).
- [61] *MicroK8s - Zero-ops Kubernetes for Developers, Edge and IoT*. URL: <https://microk8s.io/> (visited on 06/26/2022).
- [62] *ML Industry Report | 2021 Machine Learning Practitioner Survey*. URL: <https://go.comet.ml/report-machine-learning-practitioners-survey.html> (visited on 06/27/2022).
- [63] *Multipass Orchestrates Virtual Ubuntu Instances*. URL: <https://multipass.run/> (visited on 06/26/2022).
- [64] *Open Source Data Science Collaboration - DagsHub*. URL: <https://dagshub.com/> (visited on 06/26/2022).
- [65] *Open Source Machine Learning at Scale with Kubernetes - MLOps Lifecycle for Data-Scientists & Machine-Learning Engineers - Model & Data Centric AI*. Polyaxon. URL: <https://polyaxon.com/> (visited on 06/26/2022).
- [66] *Red Wine Quality*. URL: <https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009> (visited on 06/16/2022).
- [67] *Setup Google Drive Remote*. Data Version Control · DVC. URL: <https://dvc.org/doc/user-guide/setup-google-drive-remote> (visited on 06/16/2022).
- [68] *Towards an Ontology of Terms on Technical Debt*. URL: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=312135> (visited on 04/13/2022).
- [69] *Valohai MLOps Platform - Train, Evaluate, Deploy, Repeat*. URL: <https://valohai.com/> (visited on 06/04/2022).
- [70] *Welcome to Click — Click Documentation (8.1.x)*. URL: <https://click.palletsprojects.com/en/8.1.x/> (visited on 06/11/2022).
- [71] *What is MLOps?* Databricks. URL: <https://databricks.com/de/glossary/mlops> (visited on 06/03/2022).