A Review on How Machine Learning Operations (MLOps) are Changing the Landscape of Machine Learning Development for Production

Ladson Gomes Silva

Trabalho de Graduação

Recife
2022
A Review on How Machine Learning Operations (MLOps) are Changing the Landscape of Machine Learning Development for Production

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: Prof. Stefan Michael Blawid

Recife
2022
To my beloved late grand-mother Guiomar, and my parents Cleonice e Luiz who always supported me through my journey and made it all possible
Acknowledgements

This is a rather difficult task to do without spending countless pages. I have been in the university for way too long and have been through hard and rewarding times with the company and the help of many people whom I am glad to call friends. I first and foremost want to thank my parents, whom we gently call Fia and Biro. They were the ones who made it possible for me to be here in the first place. They got through some tight times to make it possible for me to live in Recife. I want to thank my grandmother Guiomar, who supported me and always said how good it would be to have a "doctor" grandson. She was a farmer her whole life, and to have grandsons in the university was her pride and joy.

To have teenagers living under the same roof, through thick and thin, and growing into adulthood together while living away from home to get a degree, in most times, the first one in the family, is a tough thing. Moreover, I am so grateful to have learned how to become "someone in life" with you and have lived these past years in the company of Ilária, Ruam, Gabriel, Geise, and Wellen.

The last few weeks have been particularly difficult, and for me to get through this was an enormous task, and I can’t thank enough my advisor Stefan Blawid for his patience, understanding, and motivation when he believed I could get it done in time more than I did myself. And talking about the difficult times at the last minute of the graduation, I could never thank enough my best friend Igor, who literally stood with me late at night on Discord while both of us were doing our thesis, supporting each other. I don’t think I would be writing it right now if it wasn’t for you.

These last two years, I’ve been blessed with working in an environment that allowed and inspired me to move forth, and for that, I would like to thank my manager and coworkers at Pitang Labs: Carlos, Massilon, Nathalia, Lucas, Anne and my former coworker Pedro Caminha who was my partner in crime for over a year.

I would like to mention one of my closest friends now, that has inspired me and made me want to be a better person every day. Someone who always nudges me to go forth and make a change and cheers all my conquests. Thank you, André.

Now coming a long way since my first semester at the university, I have the most wonderful friends someone could ever hope to have. We have built a lot together in the university and outside friendships to last for life. Thank you for being there for me all the time and literally pushing me through some of the most challenging times, Laís, Mariana, Júlia, Gabriela, and Thiago <3

And for the last section of acknowledgments, there is no way I could ever thank you enough, professor Edna Barros, for literally sitting down with me and helping me get through the university when I was stuck and lost. She has been a wonderful person in my life. Even though
she was not my actual teacher in a class, she was my mentor for most of my graduation. And she was the person behind the ESTUFA, the laboratory where I was there to see it being born. Beginning when we went out to participate in the embedded systems competitions with her. And my time in that Lab has been one of the most rewarding times of my graduation, where I’ve learned a lot, traveled, had fun, grown as a professional, and, most important of all, made many friends that I keep carrying with me wherever I am. There were a lot of people there, and almost everybody I mentioned so far was part of it too, but I want to name a few who helped me through my journey and have made it more pleasant to be there: Sara, Rodrigo Valença, Augusto, Matheus Ferreira e Mari, Hugo e Samuel, Deodato, Job, Rafael Vasconcelos, Tiago, Carriço, Marinho, Luana, Antônio Netto, Gabriel Matheus, Gabriel Marques, Italo, Erick, Lea, Casa Nova, Mino, Farias, Levi, Caio, Barney and a lot of other people I might have forgotten to mention. Thank you all <3
Life happens wherever you are, whether you make it or not.

—UNCLE IROH (Avatar: The Legend of Aang)
Resumo

Sistemas de Aprendizagem de Maquina (AM) têm sido um tema muito popular nos últimos anos e muito foi discutido e pesquisado sobre como criar os melhores modelos de AM. Mas, apenas recentemente se iniciaram as discussões sobre os desafios de colocar um sistema de AM em produção. As Operações em Aprendizado de Máquina, do inglês, Machine Learning Operations (MLOps), surgiram como um meio de estender o DevOps para atender às necessidades de juntar as atividades de operação e desenvolvimento de modelos de Aprendizagem de Máquina. O presente trabalho discute como o MLOps emergiu para preencher as necessidades de automação no desenvolvimento de aprendizado de máquina e como ela pavimentou o caminho para que as empresas e os desenvolvedores pudessem criar soluções com AM para seus produtos em pouco tempo. Este trabalho examina as ferramentas e plataformas que foram desenvolvidas e como elas tornaram possível reduzir o tempo de desenvolvimento usual para sistemas de AM e ferramentas alimentadas por IA para serem desenvolvidos e colocadas online em produção. Além disso, foram discussos possíveis caminhos a serem traçados para o futuro.

Palavras-chave: Aprendizagem de Maquina, Inteligencia Artificial, MLOps, DevOps, Ferramentas de IA
Abstract

Machine Learning systems have been a hot topic for the past few years, and a lot has been discussed and researched about how to make the best ML Models, but only recently emerged the challenges of putting an ML system on production. Machine Learning Operations (MLOps) surged as a way to extend Developer Operations (DevOps) to fit the machine learning necessities of unifying its development and operations. This work discusses how MLOps has emerged to fill the need for automation in Machine Learning development and how it has paved the way to enable companies and developers to create solutions with ML for their products in no time. This work looks at the tools and platforms that have been developed for MLOps, and how they have made it possible to shorten the usual time for ML systems and AI-powered software artefacts to be developed and deployed online in production.

Keywords: MLOps, DevOps, AI-Powered tools, AutoML, Machine Learning, ML Tools
CONTENTS

4.3.3 Budget 22

5 Conclusion 23
  5.1 Future developments 23
    5.1.1 AI Hub & Pre-built pipelines 24
# List of Figures

1.1 Timeline of ML development  
2.1 Elements of a ML system. Source: [8]  
2.2 MLOps Life-cycle. Source: [12]  
2.3 MLOps Pipeline Overview  
2.4 Model = Data + Code. Source: [17]  
3.1 Vertex AI Architecture  
3.2 Vertex AI AutoML Architecture  
4.1 Vertex AI Experiment Architecture on Pipelines  
4.2 Vertex AI Workbench for creating a Notebook  
4.3 AutoML Image training code for Vertex AI  
4.4 Vertex AI Dataset with versioning of dataset  
4.5 Vertex AI Model Registry  
4.6 Vertex AI Endpoint Registry  
4.7 Vertex AI Endpoint Sample Request  
4.8 Vertex AI Experiment Version Details  
4.9 Vertex AI Evaluate the experiment  
4.10 Vertex AI Experiment Confusion Matrix  
4.11 Vertex AI Online Prediction with the example image of a rose
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Data Preprocessing tools. Based in [12]</td>
<td>9</td>
</tr>
<tr>
<td>3.2</td>
<td>Model development tools. Based in [12]</td>
<td>11</td>
</tr>
<tr>
<td>3.3</td>
<td>MLOps and ML Deployment Platforms. Based in [12]</td>
<td>12</td>
</tr>
<tr>
<td>3.4</td>
<td>AutoML tools and platforms. Based in [12]</td>
<td>14</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Society has faced many disruptive transformations that have fundamentally changed the way we do things. The pace of change enables new technologies to arise by expanding the limits of our knowledge. And just like the introduction of electricity empowered factory work to go beyond what was possible during the industrial revolution, the internet enhances the capabilities of what we can accomplish with a single computer.

The transition from paperwork and bureaucratic operations to computer-based operations in industry boosted the digital transformation. And with the advances in digital transformation [1], online applications have come from being a necessity only for big companies to being a must for any digital product and company.

But to keep an application online in one’s environment is quite complex and costly. This challenge led to the growth of third-party services that host online applications and the rise of public cloud providers [2]. It is essential to mention that a public cloud provider does not necessarily offer free services; instead, the word public merely means that anyone can use its cloud services to host an application.

Today, cloud providers cover a diverse set of continuously increasing services, including computing services, data storage, networking, and artificial intelligence services. Today’s leading public cloud providers are Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure.

As software engineering has advanced, the massive use of cloud computing has helped popularize the use of agile methods for software development. This trend gave birth to developer operations, aka DevOps [3]. DevOps is a way of doing software development in which the work of shipping code to the cloud is no longer a task for the developer but for a pipeline of services that runs automatically. DevOps employ the principles of continuous integration (CI) and continuous deployment (CD). CI of services and code developed by different individuals enables automatic code testing and improvement on the go, catching bugs before the code is shipped. CD is about code shipment to the cloud after test and integration. Today, every major cloud provider has a DevOps pipeline to help with implementation, compatible with other open-source tools that we can easily find [4]. The availability of such pipelines transformed the way we build and deploy code, with significant gains in productivity and reliability of the code that goes to production.

Currently, applications using artificial intelligence (AI) and machine learning (ML) are becoming popular in many digital products. Companies have started realizing the value of AI and ML approaches and deploying them to production. But developing, testing, deploying, and improving such digital products is more complex than traditional software development, e.g., for a web page or mobile app [5]. ML systems are composed of three different artefacts subject
1.1 WHERE DO WE COME FROM?

We can draw a timeline through the decades to highlight distinct phases in the development of machine learning [7], as seen in Figure 1.1. As machine learning advanced in the 1990s through research in different fields, the computational power was still expensive and limited to research. At the beginning of the 2000s, big companies started investing in ML to power their products, such as Netflix’s recommendation system. However, ML was still too expensive to be used daily for small companies.

In the 2010s, fueled by digital transformation, the growth of cloud platforms, extensive research, and dropping prices for computational power, ML started to grow to its full potential. These favorable circumstances allowed small companies and developers to use ML in their products. However, with the growing diffusion of ML, the challenges for small companies to keep improving and serving ML online started to appear.

In the 2020s, with the acceleration of digital transformation caused by the Covid-19 pandemic, most companies started shifting part of their work to the cloud, relying on ML techniques to improve their work. Machine learning as offered service became popular. Whole new platforms rose, aiming to facilitate the implementation, the deployment, and monitoring of productive ML services.

Figure 1.1 Timeline of ML development
1.2 Objectives

This work aims to give the reader insights into machine learning operations, how they work and how to apply them. Mapping and classifying the current main tools, approaches and existing platforms for MLOps and ML development will help the reader get started. After gaining a better understanding of the tools, a practical implementation example guides the reader into the practicality of working with MLOps.

1.3 Outline

The present work is organized as following:

1. Chapter 2 - What are MLOps?: The chapter defines MLOps in a formal way and gives a better understanding of what they are and how they work.

2. Chapter 3 - Tools and Platforms: The chapter highlights tools and provides insights about what is out there to help make ML development faster and easier.

3. Chapter 4 - Demonstration on Google Cloud Platform: The chapter presents an implementation of an ML project using Google Cloud’s tools for ML development with MLOps.

4. Chapter 5 - Conclusion
CHAPTER 2

What is MLOps?

The most common misconception about building a Machine Learning (ML) application is that getting the model to work is the most challenging part. Actually, in real life, constructing a functional ML model is just a small part of a much bigger picture as shown in Figure 2.4.

After collecting data, verifying, analyzing, extracting, getting the machines ready, and finally training the model, the challenge of moving it to production comes, where you will need to scale a serving infrastructure, monitor, and manage your model all the time. That’s where the necessity of MLOps begins [9].

2.1 Definition

Machine Learning Operations (MLOps) is a term that refers to the set of practices and concepts needed to develop and operate Machine Learning (ML) projects [10]. The goal of MLOps is to guide teams through the challenges of getting ML systems to production.

2.1.1 ML + Dev + Ops

As one might perceive, MLOps inherits its name and some of its core principles from DevOps. It has some specific unique challenges because the life-cycle of MLOps manages not only code but also data and models. The life-cycle of MLOps can be described as the unification of the known cycles of Dev and Ops with the development cycle of Machine Learning, as shown in Figure 2.2. And as DevOps employ Continuous Integration (CI) and Continuous Delivery (CD), MLOps extends it to the third principle of Continuous Training (CT) [11].
Continuous Training (CT) implies that models must be continuously retrained when a specific event happens, such as model degradation or the availability of new training data. Then, by applying DevOps practices to ML applications, we create MLOps to describe the combination of development (DEV) and operations (OPS) in an ML system.

![MLOps Life-cycle](image)

**Figure 2.2** MLOps Life-cycle. Source: [12]

### 2.1.2 Maturity Levels

The MLOps system can be classified on the level of automation, or as known in the community, maturity levels [13]. There is no absolute model for maturity levels, and different companies have classified it differently, but for the moment, we are taking a look at the Microsoft model [14]. The Microsoft MLOps maturity model presents five levels of maturity:

- **Level 0** – No MLOps: All the processes are manual.
- **Level 1** – DevOps but no MLOps: There is automated builds and tests for application code, but limited feedback on how well a model performs in production
- **Level 2** – Automated Training: Releases are manual, but the training environment is fully managed
- **Level 3** – Automated Model Deployment: Entire environment managed, full traceability and automated tests for all code.
- **Level 4** – Full MLOps Automated Operations: Full system automated and monitored, production systems are providing information on how to improve, or automatically improving with new models. It is approaching a zero-downtime system.

### 2.2 MLOps Pipeline

The development and deployment of an ML application can be more tricky than simply collecting data, training a model, and deploying it to get predictions. Connecting all the parts
involved in this process and orchestrating it can be a real challenge because ignoring system maintenance may lead to significant technical debt [8].

Figure 2.3 MLOps Pipeline Overview

2.2.1 Data Preparation

Data preparation is the process of gathering, combining, structuring, and organizing data enabling its use in different everyday applications. Data preparation includes preprocessing, profiling, cleaning, validation, and transformation. Frequently, data gathering from various internal systems and external sources is required. The goal is to obtain data as clean as possible for possible manipulations and creation of models [15].

Data processing becomes a crucial part of any ML project, as the generated data are model inputs and directly influence the model training and its results.

2.2.2 Model Development

In the most simple words, a model is a resulting artefact containing the weight values for a neural network after training it through a machine learning algorithm on top of a given set of data. Models are created through training, using a labeled set of data [16].

Figure 2.4 Model = Data + Code. Source: [17]

The model "learns" with the data, which means that the model adjusts its weights, aiming to classify the data set accurately. The better the data used to train the model, the better the model will be at generalizing to new, unseen data.
2.2.3 Deployment

Once a model has passed the requirements for production, it is time to deploy. Deploying a model is nothing more than hosting it online, so other services can call it to make predictions either online as an API, or in batch by sending a load of data to be predicted. The model is hosted either on a physical device on-premises or a machine in the cloud. There is considerable infrastructure involved in keeping a model online. If the model is not well documented and code versioning enforced, it may lead to substantial technical debt, as shown in Figure 2.4.

Many papers show how to build models and write ML code, but only a few talk about the hidden infrastructure involved in keeping the model online.

One essential task about deployment is to monitor how one’s system is behaving and to take action in case something happens to prevent the service from going down. There are many techniques for keeping a system alive in traditional software engineering, and there is a whole discipline on Site Reliability Engineering (SRE). Though most are applicable to online ML systems, some problems, such as model decay, call for different approaches.

As will be detailed in the section about Continuous Training, a model’s performance tends to fade with time, and it becomes necessary to monitor a model’s performance to trigger re-training when needed.

2.3 Principles

The principles of MLOps serve as guides for practical implementation. They refer to how things should behave rather than how you should implement them, so it is more of a reference than a rule since each project has its own needs. Three main principles describe what MLOps is about: Continuous Integration, Continuous Deployment, and Continuous Training.

2.3.1 Continuous Integration

Continuous Integration (CI) enables an organization to have short-term and frequent software release cycles. It aims to improve software quality and team productivity and guarantees recurrent integration and merging of new code, usually multiple times a day. For doing so, automating software building and testing is essential [18].

2.3.2 Continuous Deployment

Continuous Deployment (CD) automatically and continuously deploys the application to production and custom environments [18]. Deployment is triggered instantly when a new version has passed through continuous integration pipelines.

2.3.3 Continuous Training

Continuous Training (CT) allows machine learning models to adapt to changes in data. The trigger for a model rebuild can be a data change, model change, or code change.
As soon as a machine learning model goes into production, the model’s performance degrades. Degradation is unavoidable since the model is sensitive to changes in the real world and user behavior. While all machine learning models decay, the speed of decline varies over time, correlated to data drift, concept drift, or both.

The best approach to deal with data drift is constantly monitoring the data with advanced MLOps tools. So in a way, continuous training is meant to mitigate the problems caused by a change in the input data, adapting to it as necessary.

2.4 AutoML

As the demand for Machine Learning has grown and companies try to integrate ML into their products as fast as possible, a solution has emerged for trying to automate the creation of Machine Learning models trying to shorten what used to take months to mere weeks or days. AutoML [19] delivers what its name suggests, automatic machine learning. It comes as a no-code model development to help ML developers speed development up without having to write custom code for everything.

AutoML enables even developers with limited knowledge of ML to train high-quality models with their data with little effort. It is best in speeding up development for generic problems on usual data types, e.g., images, tables, text, and video. However, AutoML might not work well if one’s business requirements are too far away from the common domains, such as classification, regression, and forecasting. Auto ML attempts to identify the best models that fit the data within a predefined set.
As the popularity of Machine Learning (ML) grows, so does the number of open-source and private tools aiming to automate the processes of using ML on products. This chapter provides an overview of such instruments, grouped by the stage of the MLOps workflow they address, as well as the platforms that facilitate the development of ML systems.

3.1 Data Preprocessing Tools

We begin with data preprocessing tools. Table 3.1 lists a number of tools for data preprocessing and data versioning. Data preprocessing includes labeling, transforming, and cleaning.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>iMerit</td>
<td>Private</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Pachyderm</td>
<td>Private</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Labelbox</td>
<td>Private</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Prodigy</td>
<td>Private</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Comet</td>
<td>Private</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Data Version Control</td>
<td>Open Source</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Qri</td>
<td>Open Source</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Weights and Biases</td>
<td>Private</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Delta Lake</td>
<td>Open Source</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Doccano</td>
<td>Open Source</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Snorkel</td>
<td>Private</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Supervisely</td>
<td>Private</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Segments.ai</td>
<td>Private</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Dolt</td>
<td>Open Source</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>LakeFs</td>
<td>Open Source</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Vertex AI Data Labeling</td>
<td>Public</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Vertex AI Dataset</td>
<td>Public</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Cloud Dataflow</td>
<td>Public</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Cloud Dataproc</td>
<td>Public</td>
<td>Data Preprocessing</td>
</tr>
<tr>
<td>Neptune</td>
<td>Private</td>
<td>Data Versioning</td>
</tr>
<tr>
<td>Git LFS</td>
<td>Open Source</td>
<td>Data Versioning</td>
</tr>
</tbody>
</table>

Table 3.1 Data Preprocessing tools. Based in [12]
Out of the tools mentioned in Table 3.1, LakeFS [20] caught my attention in Data Versioning. It is an open-source platform that works in a Git-like manner with branching, committing, and merging data that scales to Petabytes using Google Cloud Storage or Amazon S3. It works in a way that allows one to manage data across different environments according to one’s own rules. LakeFS can revert data changes, providing consistency for avoiding cascading quality issues.

For Data Preprocessing, I would like to highlight Cloud Dataflow [21], a managed Apache Beam service on the Google Cloud Platform. It can set up an Apache Beam cluster on the cloud in a few minutes, totally managed, leaving you only the job of actually operating it. Cloud Dataflow excels at dealing with data streams and processing them in a parallel environment. For using it, there are already a lot of open-source templates where you can plug in your data source and program the rules and transformations to be applied. Cloud Dataflow takes care of the rest. The tool can easily integrate with data pipelines and lakes. It automatically escalates horizontally, which means that if your data stream grows near its maximum capacity, Cloud Dataflow can raise new instances for handling the extra load and not letting the service down.

Data labeling is one of the required data preprocessing tasks when you’re collecting your raw data. Data labeling can be time-consuming and, in some cases, has to be done manually by humans. For this task, Vertex AI has a tool called Data Labeling that allows a team of humans to label your data from your instructions, preparing the dataset for training a machine learning model.

And as we can see in the table, there are a lot of other tools aiming to perform those mentioned services, and more have been surging in the last years.

### 3.2 Modeling Tools

There are many modeling tools available. Table 3.2 shows some of the most popular tools classified for three main uses: Hyperparameter Optimization, Feature Engineering, and Experiment Tracking.

Hyperparameter Optimization tools automate the search for the best hyperparameters of an ML model, including the size of a neural network, activation functions, and types of regularization that could be adjusted to get different models [22].

Feature Engineering tools help speed up the process of feature extraction from raw data to create optimal training data [23]. This process can improve performance and accuracy by removing features that are not important, choosing symbolic values to represent information, or creating new features. For example, adding the day of the week to a date can provide better insights throughout the week.

Experiment Tracking tools monitor the version of training data, hyperparameters, and results of each experiment, allowing for comparisons of different experiments.

Out of the Hyperparameter Optimization tools in this list, the one I have used the most was Optuna [24]. Optuna is simple to use and compatible with any deep learning or machine learning frameworks, such as TensorFlow, Scikit-Learn or PyTorch. Optuna parallelizes hyperparameter searches over multiple threads without modifying the code.

For Experiment Tracking, MLFlow [25] has been one of the most popular tools in the com-
3.3 MLOPS PLATFORMS

As the need for platforms and tools to integrate the pipelines of MLOps has emerged, some big companies were the first to have that need and act on it. Right now, there are a lot of different platforms that aim to be end-to-end on the lifecycle of ML and others that focus on

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperopt</td>
<td>Open Source</td>
<td>Hyperparameter Optimization</td>
</tr>
<tr>
<td>SigOpt</td>
<td>Public</td>
<td>Hyperparameter Optimization</td>
</tr>
<tr>
<td>Iguazio Data Science Platform</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>TsFresh</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>Featuretools</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>Comet</td>
<td>Private</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>Neptune.ai</td>
<td>Private</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>TensorBoard</td>
<td>Open Source</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>Google Vizier</td>
<td>Public</td>
<td>Hyperparameter Optimization</td>
</tr>
<tr>
<td>Scikit-Optimize</td>
<td>Open Source</td>
<td>Hyperparameter Optimization</td>
</tr>
<tr>
<td>dotData</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>Weight and Biases</td>
<td>Private</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>CML</td>
<td>Open Source</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>MLFlow</td>
<td>Open Source</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>Optuna</td>
<td>Open Source</td>
<td>Hyperparameter Optimization</td>
</tr>
<tr>
<td>Talos</td>
<td>Open Source</td>
<td>Hyperparameter Optimization</td>
</tr>
<tr>
<td>AutoFet</td>
<td>Open Source</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>Feast</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>GuildAi</td>
<td>Open Source</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>Rasgo</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>ModelDB</td>
<td>Open Source</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>HopsWork</td>
<td>Private</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>Aim</td>
<td>Open Source</td>
<td>Experiment Tracking</td>
</tr>
<tr>
<td>Vertex AI Feature Store</td>
<td>Public</td>
<td>Feature Engineering</td>
</tr>
</tbody>
</table>

Table 3.2 Model development tools. Based in [12]

munity. It not only has an experiment tracking functionality, but it is also a whole machine learning lifecycle platform. MLFlow is open source and can be deployed anywhere. It currently offers four components: MLflow Tracking, which is responsible for recording and query experiments looking at code, data, config, and results; MLflow Projects, which packages the data science code to run on any platform; MLflow Models, which deploys machine learning models in serving environments; and Model Registry for storing, discover and manage models in a central repository. All those features in an open-source platform striving in the community have made it a success for MLOps.
Model Monitoring or Model Deployment/Serving. On Table 3.3 we can see a list of tools and platforms to do just that.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Cloud Platform</td>
<td>Public</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Microsoft Azure</td>
<td>Public</td>
<td>end-to-end</td>
</tr>
<tr>
<td>H2O.ai</td>
<td>Open Source</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Unravel Data</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Algorithmia</td>
<td>Private</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Iguazio</td>
<td>Private</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Databricks</td>
<td>Private</td>
<td>end-to-end</td>
</tr>
<tr>
<td>TensorFlow Serving</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Featuretools</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Amazon SageMaker</td>
<td>Public</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Kubeflow</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>OpenVino</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Triton Inference Server</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Fiddler</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Losswise</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Alibaba Cloud ML Platform for AI</td>
<td>Public</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Mlflow</td>
<td>Open Source</td>
<td>end-to-end</td>
</tr>
<tr>
<td>BentoMl</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Superwise.ai</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>MLrun</td>
<td>Open Source</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>DataRobot</td>
<td>Private</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Seldon</td>
<td>Private</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Torch Serve</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>KFServing</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Syndicai</td>
<td>Private</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Arize</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Evidently AI</td>
<td>Open Source</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>WhyLabs</td>
<td>Open Source</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Cloudera</td>
<td>Public</td>
<td>end-to-end</td>
</tr>
<tr>
<td>BodyWork</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Cortex</td>
<td>Private</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Sagify</td>
<td>Open Source</td>
<td>Model Deployment/Serving</td>
</tr>
<tr>
<td>Aporia</td>
<td>Open Source</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Deep checks</td>
<td>Private</td>
<td>Model Monitoring</td>
</tr>
<tr>
<td>Google Cloud Vertex AI</td>
<td>Public</td>
<td>end-to-end</td>
</tr>
<tr>
<td>TensorFlow Extended</td>
<td>Open Source</td>
<td>end-to-end</td>
</tr>
</tbody>
</table>

Table 3.3 MLOps and ML Deployment Platforms. Based in [12]

There are a lot of different tools and platforms out there that provide managed services for
open-source software.

### 3.3.1 Most Popular Platforms

Many lists classify the best MLOps platforms [26]. From a popularity perspective, there are three dominant ones:

- Amazon SageMaker
- Azure Machine Learning
- Google Cloud Vertex AI

These are the most popular ones, simply due to their practicability, being attached to the largest cloud platforms used by millions. They all are end-to-end platforms that manage the whole life cycle of machine learning. But behind the managed platforms, they all run tools like Kubeflow, MLflow, or TensorFlow Extended to manage the MLOps pipelines.

All AutoML platforms make it easier to develop ML applications offering a managed infrastructure and state-of-the-art tools to support the ML live cycle. The use of known open-source frameworks avoids lock-in. Azure, for example, offers a drag-and-connect tool for tasks like cleaning and testing machine learning data, which makes it very easy for beginners. On the other hand, users familiar with Jupyter Notebooks might prefer SageMaker and Vertex AI.

All of the mentioned platforms offer initial development funding, which helps to explore the tools. But beware of the costs when keeping a model online. Fees are charged by the hour, and the machines can get expensive if not cautiously used.

Due to our familiarity with Google Cloud, we further explore Vertex AI and chose the platform for the AutoML demonstration.

### 3.3.2 Vertex AI

Vertex AI is an instrument that allows users to manage and train models for machine learning. The tool relies on Google Cloud’s infrastructure, which makes it easy to integrate with other project parts. Vertex AI is flexible and can be customized to fit user needs. For example, users can train a custom model, use the AutoML-trained model, or even upload a model trained previously elsewhere and still be able to use the model registry to deploy it online or make batch predictions.

Vertex AI also allows users to manage the data preparation phase by versioning, labeling, and taking advantage of the Feature Store. Users can also go through the model development phase and track the training and development of the model before deploying it.

One last feature in Vertex AI that catches my attention is the functionality to use Explainable AI for some models, improving the understanding of why the model is giving the output it is and what is influencing its decision.
3.4 AutoML tools

In AutoML tools, we have a few instruments that specialize in providing AutoML services listed in Table 3.4.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-sklearn</td>
<td>Open Source</td>
<td>Tool</td>
</tr>
<tr>
<td>Auto-Keras</td>
<td>Open Source</td>
<td>Tool</td>
</tr>
<tr>
<td>TPOT</td>
<td>Open Source</td>
<td>Tool</td>
</tr>
<tr>
<td>Auto-Pytorch</td>
<td>Open Source</td>
<td>Tool</td>
</tr>
<tr>
<td>BigML</td>
<td>Private</td>
<td>Tool and Platform</td>
</tr>
<tr>
<td>Google Cloud Vertex AI AutoML</td>
<td>Open Source</td>
<td>Platform</td>
</tr>
<tr>
<td>Akkio</td>
<td>Open Source</td>
<td>Platform</td>
</tr>
<tr>
<td>H2O</td>
<td>Private</td>
<td>Platform</td>
</tr>
<tr>
<td>Microsoft Azure AutoML</td>
<td>Private</td>
<td>Platform</td>
</tr>
<tr>
<td>Amazon SageMaker Autopilot</td>
<td>Private</td>
<td>Platform</td>
</tr>
</tbody>
</table>

Table 3.4 AutoML tools and platforms. Based in [12]

These tools help to incorporate machine learning into products relatively fast. Developers can build a proof of concepts and small applications empowered by ML with just a few lines of code. Out of those tools, we have a deeper look at Vertex AI AutoML, Google Cloud’s
3.4 **AUTOML TOOLS**

AutoML tool.

### 3.4.1 Vertex AI AutoML

Vertex AI AutoML is the tool inside Vertex AI that implements AutoML. It was formerly an independent service on Google Cloud but migrated to Vertex AI, see Figure 3.2. AutoML is quite simple to understand: After one’s initial inputs on the nature of the problem, AutoML automatically goes through Google’s model zoo to look for the model that would be a better fit for your case. The algorithm and training methods will change based on the use case and data type. This procedure is required since many different subcategories of machine learning exist, working with different constraints and problems.

The performance of a model is just as good as the quality of the data used in training. Thus, one needs to make sure to include enough data relevant to the specific use case.

![Figure 3.2 Vertex AI AutoML Architecture](image-url)
To demonstrate that it is possible to deploy a functional ML model online and go through a training pipeline using MLOps tools, we conducted experiments to get from the data to an online model serving on an API endpoint in less than four hours of development.

For this, we used the tools on Vertex AI for building an Image Classifier with AutoML on the Vertex Pipelines, with the whole system defined on a Jupyter Notebook running on Vertex AI Workbench following a Python Notebook on the Google Cloud official sample repository [27]. We used the Flowers dataset from the TensorFlow Datasets stored on a public Cloud Storage bucket. The model predicts five species of flowers: dandelion, daisy, sunflower, tulip, and rose.

### 4.1 Pipeline Architecture

The experiment used Vertex AI Pipelines for orchestration, so we could see the components of the pipelines, their output, and information about the execution of each step and how they connect. Figure 4.1 depicts the experimental pipeline from top to bottom. First, the pipeline creates the endpoint and image dataset in parallel. After finishing the dataset version, AutoML starts and automizes the model training on the image data. And once the training had generated a model, it could trigger the deployment to the endpoint allowing access to the model.
4.2 Step by Step

Let us follow step-by-step the experimental pipeline for a better understanding of the required tasks. The very first step is to get on Google Cloud Vertex AI.

4.2.1 Set Up the Environment

The first thing to be done is to create a Jupyter Notebook on the environment of TensorFlow. As in figure 4.2, you have to be on the Workbench tool on the left menu and then click on the New Notebook button to create a new notebook.

![Figure 4.2 Vertex AI Workbench for creating a Notebook](image)

4.2.2 Defining and Running the Pipeline

Once the environment is all set, it is time to set up the pipeline and define the training job. Since we are using AutoML, the whole ML training code simplifies to what we can see in Figure 4.3. The remaining code can be seen on the repository [27]. After creating the notebook, you could begin by importing one of the notebooks on the repository [27], getting the service account, and activating the necessary APIs.

```python
training_job_run_op = gcc_aip.AutoMLImageTrainingJobRunOp(
    project=project,
    display_name="train-automl-flowers",
    prediction_type="classification",
    model_type="CLOUD",
    dataset=ds_op.outputs["dataset"],
    model_display_name="train-automl-flowers",
    training_fraction_split=0.8,
    validation_fraction_split=0.2,
    test_fraction_split=0.2,
    budget_milli_node_hours=8000,
)
```

![Figure 4.3 AutoML Image training code for Vertex AI](image)
4.3 Results

The results of this experiment can be merged on the quality of the model done with AutoML on an MLOps pipeline in under three hours of getting the code ready for the dataset, running, and training it. Let us take a look at the artifacts generated by the pipeline.

4.3.1 Artifacts

The first artifact is the dataset versioning seen in Figure 4.4. Then comes the registration on Model Registry as in Figure 4.5. For deployment, we register the model to the Endpoint Registry as seen in Figure 4.6. This endpoint version can already handle online prediction requests or batch prediction. Figure 4.7 shows a sample request. The deployed application provides a REST API, through which requests can be channeled in any given way, and a Python API to assist with integrating apps developed in Python.

![Figure 4.4 Vertex AI Dataset with versioning of dataset](image)

![Figure 4.5 Vertex AI Model Registry](image)
4.3 RESULTS

Figure 4.6 Vertex AI Endpoint Registry

Sample Request

REST

You can now execute queries using the command line interface (CLI).

1. Make sure you have the Google Cloud SDK installed.
2. Run the following command to authenticate with your Google account:
   
   ```
   $ gcloud auth application-default login
   ```

3. Create a JSON object to hold your image data. Your image data should be a base64-encoded string.

   ```
   "instances": [ {
       "content": "YOUR_IMAGE_BYTES"
   }],
   "parameters": {
       "confidenceThreshold": 0.5,
       "maxPredictions": 5
   }
   ```

4. Create environment variables to hold your endpoint and project IDs, as well as your JSON object.

   ```
   $ ENDPOINT_ID=6396101031331102728
   PROJECT_ID=tg-mlops
   INPUT_DATA_FILE="INPUT-JSOn"
   ```

5. Execute the request.

   ```
   ```

Figure 4.7 Vertex AI Endpoint Sample Request
4.3 RESULTS

4.3.2 Experiment details

After the AutoML pipeline automatically created and registered all artifacts with almost no code, we assessed the modeling results. Let’s keep in mind that the results are obtained exclusively with AutoML in less than three hours of development plus training.

The experiment used 3667 pictures and took eight processing hours of training in 2h13m real time, as seen in Figure 4.8. Figure 4.9 summarizes the obtained performance metrics of the model, including the individual precision for all the types of flowers and the average value of 97.7%. The confusion matrix shown in Figure 4.10 confirms the good results. Figure 4.11 depicts an use case for the online ML model.

![Figure 4.8 Vertex AI Experiment Version Details](image-url)

---

**Figure 4.8 Vertex AI Experiment Version Details**
4.3 RESULTS

Figure 4.9  Vertex AI Evaluate the experiment

Figure 4.10  Vertex AI Experiment Confusion Matrix

Figure 4.11  Vertex AI Online Prediction with the example image of a rose
4.3 RESULTS

4.3.3 Budget

For running and training the model and all the cloud resources involved, there was a billing of only R$ 0.59.
Chapter 5

Conclusion

This work gave insights into MLOps and how they emerged to fill the need for automation in machine learning development. MLOps extend the core principles and functionalities of DevOps. The automation of ML development and operations empowers companies and developers to create solutions with ML for their products in no time, as we saw in Chapter 4.

The development of tools and platforms for addressing the necessities for achieving MLOps has been very prolific. Now we are experiencing considerable growth on some of those tools as platforms for the complete life-cycle development of ML systems, such as Vertex AI. AutoML and other ML tools as services have made it possible to considerably shorten the development time of ML systems and AI-powered applications. In the next few years, we expect to see the mass use of ML systems and AI-powered tools. To fuel the trend, tools such as low/no-code ML, AutoML, and ML services running on MLOps platforms are a must. The introduction of pipelines as a framework available in free repositories will accelerate the growth.

AutoML will empower developers with little knowledge of ML to create well-performing models with their data in no time. AutoML is vital for developing digital products once there is a severe shortage of ML specialists to satisfy the demand. Nevertheless, there are also shortcomings. An AutoML system has to try almost every model in its tool belt to encounter the best one, which consumes a substantial amount of computational power. Since the developer does not control the model choice nor explicitly knows the selection criteria, AutoML may worsen the understanding of the problem solution. The developed models may be limited to specific problems and only poorly generalize to data changes.

Furthermore, for a final thought on self-sustainable and self-evolving ML systems, the idea of an ML system that monitors and improves itself makes me wonder what more does it take to have an ML system that behaves like a living organism?

5.1 Future developments

For future developments, we can see the rise of some new techniques and services that are already popular and some that might become a reality and are worth exploring.

Some of the most popular services empowered by ML are fully trained APIs ready to use. Among those services are: Speech-to-Text, Text-to-Speech, Object detection, Image Classification, Object tracking, and more.

Another future development to look out for in the coming years is the possibility of MLOps for quantum computing. With the rise of research on quantum algorithms for machine learning and the reality of quantum computational power available in the cloud, it seems a matter of
time before it becomes popular or proves otherwise.

5.1.1 AI Hub & Pre-built pipelines

AI Hub [28] is already a reality for Google Cloud resources on AI. It is a hub for open-source ML projects and pipelines, and people can upload their own. It works as a marketplace of pipelines and ML projects. It is particularly interesting for companies that might work with AI across different teams because it allows you to share your ML pipelines for reuse. It facilitates the deployment of an MLOps pipeline with your data without having to recreate the whole MLOps pipeline.
Bibliography


