TÉCNICO LISBOA

Smart-ML - Reduce Running Costs of a ML Model in Production

João Manuel Ginja Ramalho

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Luís Manuel Antunes Veiga
Dr. Pedro Patrício

Examination Committee

Chairperson: Prof. Nuno Miguel Carvalho dos Santos
Supervisor: Prof. Luís Manuel Antunes Veiga
Member of the Committee: Prof. João Nuno De Oliveira e Silva

October 2024

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

One of these lives, I will make things right
With the wrongs I've done, that's when I unite
With the Father, Son, until then, I fight

- Kendrick Duckworth

Acknowledgments

A thesis work is never an easy task, especially since it can become a solitary task, but a lot of people helped me along this beautiful journey. I will forever remember the learnings from this work, the experiences and the hardships overcome to convey it, but especially, the people.

To my Supervisors, Professor Luís Veiga and Pedro Patrício. To Professor Luís for taking on the challenge of supervising and guiding me with this work, and for doing it effortlessly and demonstrating compassion, commitment and guidance towards the task at hand. To Pedro, for taking a new role in his eventful life as a supervisor for this work, for always providing great guidance, and great advice for life in general. I will always cherish our talks, and the superstition of Benfica always scoring the moment I ask questions during their games.

To my colleagues at FST Lisboa, during my two year stay at the team I've come across incredible people and spent wonderful moments. A heartfelt thank you to all the people that composed the Autonomous Systems department during those two years: Manuel, David, Filipe, Diogo, José, David and Rui. To a group of friends inside the team, which we self-entitled *rapipi*, we had a lot of great a fun times the past year, and without them our sanity would certainly not be kept in check, much love to all of you.

To all my friends, from university and from my hometown, you are forever in my heart for this whole journey. The times we spent doing projects until late night, studying, having coffees and going out to have a couple of drinks will be always memorable. A special thank you to the people who were there for me when I wasn't the sharpest and in the best version of me, who really listened to me and helped me get back up. To Carolina, Diogo, Guilherme, Tomás and Paulo Sérgio.

Lastly, but certainly most importantly, to my whole family. To my parents for raising me and molding me into the person I am today, for allowing me to accomplish this feat. To my uncle and aunt for always being present in my life and raising me as a son of their own, and taking me on during these years of university. To Pedro and Ana, although not family by blood, I will always cherish you like close family. To my Mother, for being a warrior her whole life and always being selfless, loving and caring towards me, I know you want me to succeed and complete this chapter of life as much or even more than I do. To my Uncle, the person who I look up to the most and is always present during my highs and lows.

To my late Grandfather.

Abstract

Nowadays technology is a crucial part of our world, present in due day life, and even cutting edge solutions being applied to several sectors of society, using new paradigms of computation that years ago would be labeled as far fetched but today are heavily used. This is the case for Machine Learning (ML) and Cloud Computing, as both of these seemed intriguing subjects in the past, but nowadays are at the disposal of every person as a service.

Companies and workplaces also take advantage of this and use cloud services as well as running ML models for various applications, although this becomes costly: running both of these services combined can deplete monetary funds pretty quickly.

In this sense, Smart-ML is presented as a way to reduce running costs of ML models in the cloud, focusing in inference. It explores techniques that balance the accuracy of a model and the cost of running it in a cloud service, as well as reviewing techniques regarding cloud computing that can be more money efficient. These are not the only factors, as more are introduced and studied more in-depth during the work. We also review use-cases of these models in different sectors and what each case requires so that we can apply something representative and according to their necessities in Smart-ML.

Keywords

Machine Learning; Production; Cloud Computing; Cost Reduction.

Resumo

Hoje em dia a tecnologia é uma parte crucial do nosso mundo, presente no quotidiano, como são o caso de soluções de vanguarda tecnológica aplicadas a vários setores da sociedade, alimentados por novos paradigmas de computação que, no passado recente, seriam rotulados de extrema complexidade e elaborada implementação - sendo hoje em dia já muito utilizados. É o caso da Aprendizagem Automática (AM) e da Computação em Nuvem, ambos intrigantes no passado, estão agora à disposição de qualquer pessoa de uma forma bastante mais simplificada *as-a-service*.

As empresas e os locais de trabalho também tiram partido deste facto e utilizam os serviços em nuvem, bem como a execução de modelos de ML para várias aplicações, embora isto se torne dispendioso: a execução combinada destes dois serviços pode esgotar os fundos monetários muito rapidamente.

Nesse sentido, o Smart-ML é apresentado como uma forma de reduzir os custos de execução de modelos de AM na nuvem, com foco na inferência. Este trabalho irá explorar técnicas que equilibram a precisão de um modelo e o custo de o executar num serviço de nuvem, tal como rever técnicas relativas à computação em nuvem que podem ser mais eficientes em termos monetários. Estes não são os únicos factores, outras técnicas também são introduzidas e estudadas em maior profundidade ao longo do trabalho. Também analisamos casos de utilização destes modelos em diferentes sectores e o que cada caso requer para que se possa aplicar algo representável e de acordo com as suas necessidades no Smart-ML.

Palavras Chave

Aprendizagem Automática; Produção; Computação na Nuvem; Redução de Custos.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

1

Introduction

Contents

In the ever-growing world of production and technology, Machine Learning (ML) started to take a preponderant position in many aspects of development, evaluation and prediction of the work industry. It is pretty recurrent in our day-to-day life to hear conversations, and even resort to using services like ChatGPT, or even learn about the rising in usage of CNNs, or object detection neural networks like You Only Look Once (YOLO)[1]. Truth is, ML has already seen plenty of usage in the industry before it broke *mainstream*. Models like Support Vector Machines were seeing already plenty of usage, as well as Natural Language Processing models, Google Translate being its prime example or even PageRank which Google still uses for its search engine. Nonetheless, Artificial Intelligence (AI) has been present in our lives for more than a decade.

The factor that played a key role for AI to be more present in our everyday life are the technological advancements that make it able to now run ML models on commercial computers, and everyone is now able to develop and explore models making this technology and studies more accessible. This allowed not only for better runtimes, but also to rapid progress in the field in the past few years. Every device that has been introduced in the past decade has been upgraded to now to able to use AI or even be *powered by AI*. Of course this also conveyed idea to run these services in third party software, as well as devices altogether, and providing AI as a service, rather than software.

With this there was a also a shift in the paradigm of how to use these type of resources: the companies are not investing in developing, testing and deploying these models themselves, and started buying already developed and tested models as services. This means that instead of buying the whole model, third party entities spend only monetary resources to utilise it as an end product. Then, they supply the data they want to run and the companies that develop the models are the ones responsible for decisions such as hosting and maintenance. Hence, the responsibilities of making sure that the profitable margins of running these models are being met lie further on the companies that develop the ML models. Given that many have found that ML models are perfect suitors to be run in the cloud and be available as a service on third party servers [2] [3, 4], the main challenge that arises is how can developers reduce costs by applying direct changes to ML models.

The relevancy of this issue comes with the growth of need of these services, as many more clever and effective ways to incorporate ML into workflows are being found, this comes with a larger resource usage and it may lead to a point where maintaining these services becomes unsustainable. This is where cost reduction comes as an important factor as it will allow for companies to allocate the resources saved elsewhere. But, while this might seem straightforward, one must take into account that the performance goals set by the company or the clients paying for the product must be met, therefore becoming a non-trivial task to obtain this balance. Adding to the fact that the product is a service in which the clients only get the outputs of a ML workflow pipeline, these services need to be up to par to the standards required

[1] https://pjreddie.com/darknet/yolo/, accessed on 10 Set 2024
[2] https://azure.microsoft.com/en-us/products/machine-learning, accessed on 8 Nov 2023.

to be employed by companies.

The main issue with how one can reduce costs, is that besides the techniques that can be applied directly to a cloud environment, the techniques can surround ML can also take a tow on the models performance themselves. What this means is that some of the most commonly used techniques are based on creating more compact models, which not only brings costly expenses to companies to produce, but also some can lead to performance decreases. When taking into account that some of the production sectors have use cases in which it is required for these models to have very high precision and accuracy scores, the work to make models more cost-effective poises a much careful and complicated matter. One great example of costly consequences can be seen in agriculture, which is starting to employ ML-models to automatize work, as it uses SVMs and ANNs for water management, and continous poor prediction from these models can lead to unnecessary water waste, and from a company standpoint, leading to a loss of profits.

While reducing costs of cloud services is not necessarily a freshly new issue, trying to make a framework with many ML models and investment on reducing each specific model has not been a study as of now, most of the work done shows what was established as a framework was to reduce cost on the cloudware itself [4, 5], or generic manipulation of how the ML models are going to be computed [6]. The work and research done towards Smart-ML is going to revolve around cost reduction, this takes into account various approaches, and considerations, such as how to to reduce costs in inference, but also what can one do in training, while considering the option for retraining (the latter due to lack of publicly accessible research and information about this topic, which can be seen in the studies mentioned in Section 3), but also understanding how much resources are used to train again a model. Besides, it also focuses on specific use cases of ML models that have different requirements of performance, or other metrics such as accuracy, F1 or even computation times. Much more factors are presented taken into account for evaluation purposes of this study of course, as they are also be presented in further Chapters.

Beyond the study of how different use cases may imply changes in the technique used, or incorporation of other techniques to reduce costs of running the model in the cloud, the approach on how to study the issue at hands is done by evaluating the most common ML models used in various fields of work and knowledge. The optimizations that are to be obtained by these techniques result in a reduction in either space, training time, or execution times in the model, or even a combination of these three. With the results obtained then a process to filter the information gathered to select the most fitted models and the relevant sectors to use the altered models in production, with the intention to formulate the most effective ways to reduce running costs.

Albeit this study strives for an effort to deliver a working framework to reduce costs of models, techniques like auto-scaling applied to Iaas, the service defined to be used for this work, can at times be the most straightforward way to be cost-efficient in the cloud, therefore we try for the techniques discovered to be an adding layer of reducing costs, so that auto-scaling can always be an assured and applied technique in production.

1.1 Current Shortcomings

The greatest issues with this work come due to the lack of documentation of the implementation of several techniques developed to have ML models become more efficient, or even the cloud frameworks created to achieve or cost-efficient usage of ML services. Therefore, what mainly matters to companies is to know how to implement these techniques that allow them to achieve fewer running costs during production. The issue that resides in most of these works present the theoretical work that backs up their premises, but it is not specified how one can implement it, leaving a lot of work to be done to reach levels of results similar to what is presented in these studies.

Some examples like MArk [4] or BATCH [5], which are frameworks that provide great insights and solutions to be applied to ML-Ops, lack the specifics on how to implement the solutions proposed, and those can be complex at times, besides not being model-oriented as well. This leaves developers with not only the task to understand the theoretical basis behind the articles, but also must convey a practical solution which meets the results found in those studies whilst not knowing the implementation details.

1.2 Contributions

In this work we present Smart-ML, a framework that contains ML-models and cost reduction techniques applied to the framework, which is presented in Chapter 3. These techniques can be centered around the models themselves, but also may not be directly applied to them, but rather a certain aspect of the whole framework in an effort to make it more cost efficient. Due to this, a Cost Model is also proposed to better understand the impact of the changes made by the techniques and to facilitate in the evaluation of each techniques.

1.3 Document Roadmap

In the length of this work we examine how one can diminish the running costs of ML models in the cloud. For this, in the Background section are given brief explanations to the current cloud computing

paradigms, as well as the state of the art of machine learning in production, not only the models themselves but their appliances. Also in the background are shown some examples of the current state of the art for ML cost reduction techniques for ML models, as well as cost-effective cloud frameworks for ML models. During the Proposed Solution chapter is presented a method to study all of the relevant ML models, and which are the most advantageous techniques to be applied to each one to become the most cost-efficient possible. In the Evaluation is presented how the testing of this study pan out and most importantly the results are displayed as well as a reflection on the results obtained and an extrapolation of them to the Cost Model implemented, and finally some closing remarks are done in the Conclusion section, along with the main takeaways of the study and future work.

2

Background

Contents

In the length of this section we go over plenty of concepts, ranging from Cloud, Big Data, Machine Learning Operations, and Machine Learning itself. The purpose for this is to grant context to this work of all the domains that revolve around it, specifying over noteworthy concepts and work on them. Some of this concepts are also used on the work, hence it is also a convenient way to introduce them ahead of their usage. A Related Systems section is also featured, to mention and briefly explain many similar systems that were developed in regards of a more efficient deployment of ML models in production. This is worthy to show as it served as great inspiration for the work done. Besides, in this section that are other systems that are not necessarily Machine Learning (ML) oriented, but when applied to it they can grant benefits to the system.

2.1 Cloud Computation

Cloud Computation is a very complex domain, but with a simple principle: it denotes a form of computation that allows an entity to have *data and services reside in massively scalable data centers in the cloud and can be accessed from any connected devices over the internet* [7]. This essentially means that it is possible to have programs, or even parts of them to be offloaded from local resources into data centers from a third party, which helps not only relieving local servers, or even personal machines, but also allows for these programs to be available worldwide, which is great in the case of companies that not only needs these type of services, but also companies that develop said programs.

For the purposes of this work is worth to go over Cloud Computation concepts to not only contextualize the reader, but also to understand deeper how it works, since the framework for this work is developed in a cloud-based system.

2.1.1 Cloud Service Provision

To be capable to access these services companies started to provide their infrastructures and services to support cloud computation, and nowadays there are several viable options available for personal or even enterprise levels of usage. Providers like Amazon (AWS) [1], Microsoft (Azure) [2], and Google (Google Cloud) [3] are some of the biggest in this market. They offer their services at a large scale, offering different types of Service Level Agreement (SLA) while all having plenty of similarities on how to use their services in a client standpoint, which is what keeps all of them correlated, and none being the clear better choice over the other.

Other important notion is that all of these services have a common ground hierarchy of services they provide to a user [1], which we go through, without delving much deeply into each concept.

[1] https://aws.amazon.com/console, accessed on 10 Nov 2023.
[2] https://azure.microsoft.com/en-us, accessed on 10 Nov 2023.
[3] https://cloud.google.com/, accessed on 10 Nov 2023.

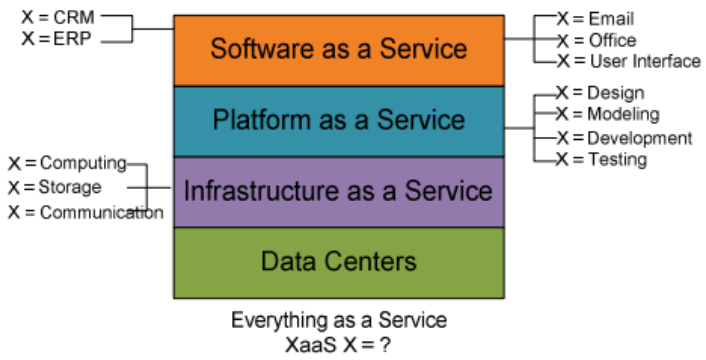

Figure 2.1: Services hierarchy [1].

Infrastructure-as-a-Service (IaaS) Built directly over the data center layer, it gives access to computing operations, data storage and connectivity, grating a user a virtualised environment with server-like capabilities [1]. These can then be subject to elasticity principles which will be mentioned further on. Moreover, IaaS is the service provided that is used for the studies conducted in the work.

Platform-as-a-Service (PaaS) Platform-as-a-Service is a layer above IaaS as it provides the user with software assistance to design, develop and maintain the desired cloud services [1]. Cloudware as it is often called, this is a great tool for distributed workspaces to have a common ground to work on, without worrying with the underlying infrastructure.

Software-as-a-Service (SaaS) Software-as-a-Service takes a step further over PaaS or the other lower layers. It is fully developed and it is used on demand. Applications like Google Maps, Slack and Zoom are prime examples of SaaS that are used on a daily basis.

Function-as-a-Service (FaaS) Function-as-a-Service comes as a recently emerging form of service for cloud computing. It comes as a form of stateless computation, that allows a user to have simple stateless functions at his disposal on demand [8], that are not consuming any resources until its execution. This last detail is very important as it is the selling point for the growing popularity of services like AWS Lambda functions, of Microsoft Azure Functions.

Besides these services, more recently have emerged new solutions to deploy and provide **ML services (MLaaS)**. These services provide users with pre-deployed ML models, as well as tools and infrastructures that facilitate one to use ML in the cloud. Tools like Amazon Sagemaker [3] are widely known to provide these type of services, and others like Tensorflow Serving proposed by Olston et al. [9] , which has the goal to create a framework which serves all the models and workloads in which the authors of this article work at. From all of this offers, for the purposes of the study we resort to IaaS.

2.1.2 Elasticity as a Cost Technique

Elasticity is a a characteristic of a cloud environment, which evaluates whether it has the capabilities to adapt to the needed workload, without ever over or under compensating [10]. Now will be presented two viable options to ensure elasticity to a cloud environment.

Auto-Scaling In the past decade auto-scaling as been studied as a cost-effective way to offer elasticity to cloud computing [11]. It can be used to scale horizontally, which scales the amount of Virtual MAchines (VM) that are running at a given moment, or vertical auto-scaling, which adjusts the amount of resources that each VM is allowed to use at a given moment (that can be done adaptively, e.g. with memory [12]). Both of these techniques are great to use to ensure elasticity in cloud computing, each having its advantages and counterparts of using it, but both are a great technique to handle workload in a very efficient manner.

Being the most consensual form of ensuring elasticity to a cloud service across several providers and over the years, auto-scaling became the standard and benchmark for what a technique like this must be able to deliver. Over recent years it was researched in many aspects to be able to fulfill much more than just efficiency in resources based on loads. Many of the major benefits from using this surround scalability, efficient resource allocation, performance improvements, and that is why this has been the state of the art for resource management for cloud services.

Nonetheless, performance is not the only key strong factor for auto-scaling: it also is a cost-efficient way of doing so. There have been many studies on how auto-scaling could even be leveraged to reduce costs, as an example one of this approaches applies Machine Learning to be able to have a better Quality of Service (QoS) or be more cost-effective (and even more energy-efficient) [13]. This is based on horizontal scaling as the authors are not sure if vertical scaling can be effective to scale Virtual Network Functions, and an ML classifier (Random Forest) to train and infer over the features, such as timestamps and traffic measurements at different times, used directly for auto-scaling.

Aside from this, managers like Kubernetes (K8S) [14] were developed by Google to provide a platform for every client and end-user to be able to easily test and utilise containerized applications in the cloud, with an end-goal to be easier to deploy these type of applications to production environments. Amazon also offers with EC2 web services that grant instance based cloud computations that have features like scalability and elastic load balancing. Likewise, Azure grants users the same features as the ones mentioned.

2.2 Machine Learning

The other major computation field that is heavily focused during the study is Machine Learning. This field is a sub field of Artificial Intelligence that allows programs to process data, and make decisions based on the given data to be able to solve specified problems. It combines knowledge from artificial intelligence, data analysis and statistics to cover plenty of domains, as it is focused on theory, performance and properties of learning systems [15]. This can then be distributed into three main types of analysis: Unsupervised Learning, Supervised Learning, and Reinforcement Learning [15, 16]. The key difference between Supervised Learning and Unsupervised Learning is that the latter does not use labeled outputs or target variables to train ML models. On the other hand, Reinforcement Learning also receives during training feedback on how it is behaving during training, in the form of a scalar that indicates how well the system is behaving at the time, not the correct action it has to take to obtain the optimal results [15, 16].

2.2.1 Models

Now a few ML models will be swiftly introduced, that involve all types of Machine Learning mentioned above. These were selected as they were the most prevalent during research that is shown in Chapter 3 of this work. Not only they are a pristine example for the work in generic terms, but also these models are very relevant in multiple sectors that are also a factor to take into account for the relevancy of the work. This does not mean that all of them were evaluated and tested, but are still noteworthy to take into account due to the different studies and systems that were studied to gain knowledge to develop the work at hand. The reasoning behind this decisions is further explained on the first section of Chapter 3, since it is a crucial part of the work done.

Support Vector Machine SVM is a supervised learning technique that is a robust data mining tool that allows data to be divided by a hyperplane. This simple yet effective model is able to be used for regression, classification and outlier detection [17]. The basic principle underlines that it should be possible to make a separation of two clusters using an hyper plane of multiple dimensions, as it can be a simple line or even a subspace. This hyper plane has parameters that can be changed to allow it to have softer margins of classification, as well as using kernel functions, which are a valuable trick to be able to not have misclassifications on points that applying soft margins principles is not the optimal solution, as it adds a new dimension to the data that is being given as input [18]. Knowing how to use kernel functions and which ones to apply to each case is very rewarding to the model as it grants the model in production performance improvements and a higher dimensional space [18].

Clustering Clustering is the primary and major technique that defines unsupervised learning, as the goal of this technique is to separate a finite unlabeled data set into a discrete set of data structures (the data is partitioned into a certain number of groups), which is different from giving a characterization to an input based on unobserved samples with the same probability distribution [19]. Most of these use similarity or dissimilarity measures to be able to compute clusters, the most used being Euclidean Distance, Manhattan Distance and Mahalanobis Distance [19].

There are several clustering algorithms, that are not necessarily better in direct comparisons, but are better suited for different cases. With this in mind we can separate them into two groups, hierarchical clustering and partitional clustering. While in hierarchical clustering data objects are grouped with sequences of partitions, in partitional clustering objects are directly clustered to be divided into clusters without any hierarchy [19]. Some examples of partitional clustering are K-means clustering, DBSCAN and OptiGrid [19].

Decision Trees Decision Trees are a popular supervised learning method of classification or regression that is based on hierarchical processing of data since they are practical and easy to understand and visualise [20], with their structure resembling much to a tree, or search trees as data structures. It has a root, branches and nodes. Until it reaches its leaf nodes, the input data, that is divided by features (can be either categorical or nominal) [20] undergoes several levels of nodes that split the data under certain criteria that is specified to meet the goals of the model, such as using Mean Squared Error for regression. After going through each node the data is split to a certain subset.

Random Forest Random Forests (RF) came as a solution to many problems that Decision Trees have, such as overfitting [21] and the growing need of processing big data which can cause an imbalanced distribution of data [22]. This ensemble ML model is made up of multiple decision trees and it is well balanced by using bootstrap sampling to keep those decision trees divided into subsets (using algorithms such as kNN to do so) [22] in an effective effort to tackle overfitting.

Isolation Forest Isolation Forests are a ML model which excel at anomaly detection. As they name implies, they are fundamentally different from other traditional Clustering models and algorithms, opposed to focusing on normal instance profiling. This proves to be more efficient in comparison to RFs and linear time distance based methods, as well as a better Area Under the Curve (AUC) score than those mentioned [23]. Overall for the specified purpose of Isolation Forests they are viable solutions to deal with large datasets, as well as having great performance with small ensembles.

Artificial Neural Network Deep learning has been gaining a lot of traction for the past decades since it became a powerful machine learning tool. ANNs try to achieve similar behaviors of a biological neural

network, and can have many appliances, as there is no golden rule to how to construct them, they can take many architectures and fulfill many goals. Its most defining trait is that it is composed of nodes that simulate a neuron [2], that are organized into features and trained, tested and executed like-so [16]. They are composed of input layers, output layers and hidden layers. Hidden layers have the responsibility of receiving the outputs from its precedent layer, make a stochastic and weighted decision on how its output will detriment or improve the final output of the data received [2]. These nodes and layers can then be orchestrated into several forms to obtain different types of neural networks, two of which are discussed. They are a pretty effective model and can help in various fields as they applications encompass computer vision, speech recognition and natural language processing [16].

Multilayer Perceptron A Multilayer Perceptron (MLP) is a type of ANN which is based on feed forward propagation, not having necessarily a sole path for information to flow, as recurring input can happen (when a neuron receives an input from a layer that is considered to be placed after it) [24]. These type of ANNs usually have a minimum of three layers: the input layer, one or more hidden layers and an output layer. Between layers, every node is connected to every node of the following layer. The hidden layers are trained using an algorithm of back propagation which involves initializing nodes with a starting weight, starting training, which involves forward propagation of signals, and its correspondent back propagation of errors and adjustments of weights, and then iterate for each epoch of training [24].

Convolutional Neural Network Frequently mentioned as CNN, this particular application of an ANN excels at image recognition and graphic related patterns. It uses a very simple architecture that comprises the input layer, convolutional layer, pooling layer, fully-connected layers, and even fully-connected layers with activation functions (primarily a rectified linear function), which then lead to outputs. The convolutional layer aims to calculate a scalar of all computations of a certain region that englobes the region of the input given to the model and the respective weights of said region, and afterwards it is applied an activation function (usually ReLu as well), to grant non-linearity. The pooling layer purpose is to downsample the spatial dimensionality of the input. The fully connected layer then acts as a classic ANN model. In some cases they may feature a ReLu, with the main goal to improve performance [2].

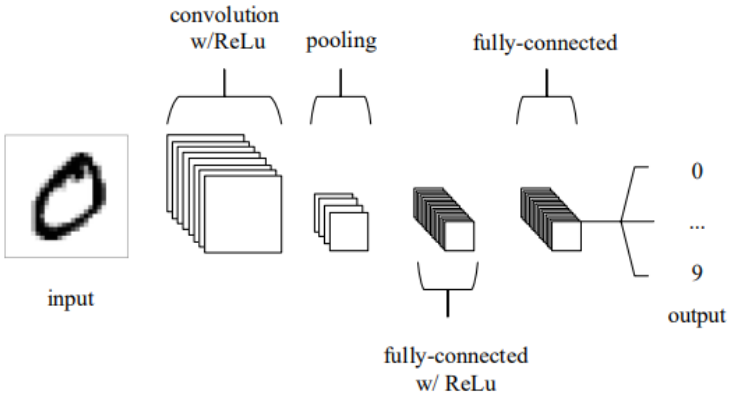

Figure 2.2: A simple CNN overview [2].

Long Short-Term Memory Recurrent Neural Networks (RNN) have become quite popular over the last years has unlike classic ANN, RNN are capable of maintaining ”memory” from previous iterations of the algorithm. This behavior makes RNN great to use for sequence-based data. A common problem these neural networks suffer from is the vainishing gradient problem, which in essence does not permit these models to maitain much long-term memory nor dependencies in their models, rendering them a loss in performance.

With this is mind Long Short-Term Memory (LSTM) was created, proposed by Hochreiter and Schmidhuber in 1997. This model introduces gates to become part of each LSTM cell, and effectively erradicate the vanishing gradient problem [25]. Since its introduction, LSTM has become quite popularized as the stand-out RNN model to use.

2.2.1.A Applications in Production

In this work we go over a minute domain of sectors that have ML models applied to them, five to be more precise: healthcare, transportation, manufacturing, agriculture and software development. The factors that were taken into account for choosing these sectors over many others were relevance to society, progress in research of ML appliances on the sector, and the depth of usage of said model in the sector (whether it is for minor tasks or do they play a bigger role in advancements made).

A – Healthcare

Despite being the sector with the least progress currently for models that are used in production, as most of its are still in research, or in highly specific levels of the field, healthcare is a sector that is seeing a lot of efforts and investigations towards how to incorporate ML into it. Plenty of direct usages like direct diagnosis of a patient to replace a trip to the hospital or emergency services are still overshadowed.

Essentially, daily clinical workflow could be improved with the usage of machine learning, with the due investment to create models to deal with these situations [26].

The more rapid progress that is being made in this is surrounding pattern identification in certain deceases, as it is known that recently studies show that the usage of AI helps to identify recent development of cancer. Most of the current techniques applied to detect early symptoms and manifestation of deceases use techniques like clustering (K-Means), but many other models are effective as well, such as CNNs, logistic regression and classic ANNs as shown in a survey made by Chugh et al. [27]. A study made by Nahid et al. [28] also displays how LSTM can be used and even that CNNs can be combined with LSTM models to increase the performance of the classification, yet the author still shares that the dataset used to test still granted problems regarding measuring performance.

B – Software Development

This area has also received a lot of investigation on how ML can become helpful for developers. There are many appliances such as Github Copilot a human pair programmer that is fairly recent and can be handy in low complexity code. But one task that ML models can give quite a lot of help to a regular code developer is software testing. In a study made by Durelli et al. [29] we can see that ANNs and decision trees are models that are quite suited to help in this subject. They facilitate in oracle generation, test-case generation, refinement and evaluation [29]. The only set back this methodology has is that a person must be careful on the quality of the data given to the models for it to create accurate models, as these need a lot of data to train and test, as well as well as assuring that testers have basic knowledge of the purpose of these algorithms, since not knowing the goals of the models will make them useless and not effective. But, is they are used correctly they grant many benefits such as little need for human intervention and support for complex testing activities [29].

C – Transportation

In terms of transportation a lot of effort has been drawn into this issue. Autonomous Driving has become a very common subject regarding transportation, and machine learning plays a critical role on the robustness and performance of the algorithms used to safely and efficiently become able to drive autonomously. A survey conducted by Grigorescu et al. [30] exposes the most prevalent ML models used for Autonomous Driving, ranging for deep learning networks such as CNNs (the famously known YOLO) for image detection, or even LSTMs that solve a lot of problems such as steering and velocity actuators or path planning/lane-keeping.

This subject is extensive, as transportation can incorporate a lot of cases, such as accident detection, parking and road anomalies. Another relevant problem that transportation suffers from is traffic. Medium and large sized cities struggle much from this issue and machine learning is helping to face this challenge by creating models that are enabling traffic predictions and route optimization. All of this can be further

examined in a study made by Zantalis et al. [31], where the authors show that using IoT as a mean to feed ML models with data to run can become very advantageous to achieve a better flow and smooth day to day of cities. During this study several models were reviewed, given it is a broad subject, such as SVM, ANN, CNNs, RF and kNN.

Also, in the study of traffic prediction Tian et al. [32] proposed the usage of an LSTM to predict traffic flow, and it achieves better results that SVM and a classic feed forward MLP.

D – Manufacturing

Manufacturing has seen a lot of benefits from adopting ML into its field in several ways to enhance tasks like supply chains, human-robot collaboration, or even employee safety. Wuest et al. [33] presents a review shows that like process optimization, monitoring, control and maintenance prediction have become much more efficient with the help of ML models. The authors even take a step and state that SVM is the better standing model to use, especially for monitoring. But aside from this, other models such as ANN and RF yield great performance results, as Paturi et al. [34] demonstrate in the review conducted by them.

E – Agriculture

This core sector to society may seem like the odd one out of all the sectors described, as it is heavily focused on nature, but even technology is finding a way to be incorporated with nature. Over the past years things like machines to control rot vegetables or even the water usage on crops, have started to gain a lot of use in production. Liakos et al. [35] did an extensive review on models used to great avail in production and yet again SVM is present in a sector, as well as ANN, since most of the work done in is crop quality control these are prime examples of models to run these tasks. Aside from this, water, soil and livestock management are also great examples of applications of SVM and ANN models. Moreover, talking specifically of CNN models, these also perform well for certain tasks to detect plant deceases or even livestock face recognition.

2.2.2 Machine Learning Operations

Machine Learning Operations, or simply **ML-Ops** [36], is to ML what Development Operations (DevOps) is to software development. It aggregates good practices that developers should follow to ensure that their models achieve the best results possible. This englobes the maintenance, continuous integration and development, as well as scalability of the systems. Not only this, but techniques such as data ingestion and transformation, as well as output presentation to the end-user are aspects that ML-Ops also takes under its responsibilities [37].

Nonetheless, aspects like sustainability, and cost-efficient ML workflows must also be aspects that should be taken into account, especially with the growth in complexity of AI systems. Tamburri [37] conveys in its article regarding sustainability towards ML-Ops, as it mentions as well that sustainability should be as important as other practices since if disregarded it can lead to saturation.

To avoid this, one needs to make sure that sustainability is maintained in ML-Ops in three different aspects: to make sure it is still operational from a technical perspective, to be able to be developed upon by companies while requiring a minimal effort and investment, and lastly be able to continuously fulfill its objective without counteracting against the contracts beforehand agreed on.

2.2.3 Big Data Processing

Big Data Processing is a very crucial matter in the running days, as information right now is much more complex and that leads to large clusters of data, as well as tasks to process it becoming more complex. It has many appliances, such as in distributed applications, and with the emerge of cloud computing, it also found its way to become useful in cloud applications.

2.2.3.A MapReduce

A technique that was first used parallel in and distributed computing applied to big data processing, it quickly was found to be a prime suitor for cloud computing to ensure more efficiency due to its simplicity and fault tolerance characteristics. It makes for a good option only for smaller companies to use when needed to process large databases [38, 39]. MapReduce is composed of two core phases that compose its name: the Map and the Reduce phase. In the map phase there are multiple processes running map functions that associate records into smaller pieces of data, which then produce a key-value pair that is used for intermediate purposes. After the data is sorted with its keys and values, it is then fed to multiple reduce tasks that process those records into even smaller samples of data, to handle and fit in memory large clusters of data [38]. Applying this with cost based optimizers can even grant better results to how much one spends while using the cloud [39]. MapReduce jobs can also be chained into workflows/dataflows executed when new relevant data becomes available, e.g. [40].

Notable evolutions dedicated to improve on MapReduce for big data processing include **Apache Spark**, which utilises it with the addition of Resilient Distributed Datasets which enables data sharing to have a bigger range of processing workloads than before, all encapsuled in the same framework [41]. **Apache Spark Streaming** then succeeded in granting a high level library that is capable of handling streaming processing by the means of micro-batching, discretized streams of events (typically tuples) and sliding window operations [42]. Then in the same note, **Apache Flink** is introduced for data processing, but with real-time and low latency capabilities, which makes it very useful for real-time data analytics [43]. Stream processing can also be applied to graphs, e.g. [44].

2.2.3.B Pandas Python

Pandas Python [4] is a Python library that is capable of data handling and data processing, which is frequently used for ML purposes. Most of the data handling is made by using its DataFrames, which are essentially two-dimentional structures which can be seen like a table with rows and columns. Unlike NumPy [5] which has the same data type for the whole data structure, Pandas is able to have a data type for each row or column of its data structure.

DataFrames have plenty of features that allow for easy data manipulation [9], since it features aspects like intuitive data alignment, advanced pivoting and reshaping, grouping and aggregating data, data visualization, as well as combining and joining data sets. Given this, it is trivial to understand why Pandas is widely used for ML development, as it facilitates processing large datasets.

2.3 Related Systems

Machine Learning embodies a predominant portion of this work, and many fields have utilised its capabilities to great extent, therefore it is now stated some similar studies done in the field of cost reduction techniques (applied in training and in inference) applied to general ML, which are relevant approaches to ML-Ops and provided great inspiration for the work to be developed. Besides these ML focused studies, it is relevant to mention that cloud computing also has had some great research towards better cost reduction. As mentioned, these systems are mentioned not for the purpose to implement them all, rather to gain knowledge on what is usually work towards making ML-Ops more efficient.

Energy-Efficient ML-Ops A study made by Desislavov et al. [45] addresses how different modelings of neural networks, such as the rapid growth of number of parameters in new neural networks can affect the energy consumption. Factors like algorithmic improvements, hardware specialisation and consumption efficiency are not the cause of increase of the energy consumption of ML models, and still maintain consumption below of what a human consumes when having the same tasks at hand. There are plenty of ways to make models more efficient, either by making the model more compact, or using some clever techniques to accomplish better cost efficiency, but there are also techniques that can be applied and increase the performance of a service not by necessarily changing the models themselves, but features surrounding it, or the framework itself.

[4] https://scikit-learn.org/, accessed on 20 Mar 2024
[5] https://numpy.org/, accessed on 20 Mar 2024

Efficient Storage Simple measures include efficient storage formats, such as Apache Parquet [6] and Apache Avro [7], which are usually used combined with Apache Hadoop [8], a distributed data storage and processing tool. While Parquet has a column-based storage and Avro a row-based, which will lead to different behaviors and therefore different purposes, both are valuable tools to utilise in Big Data systems [46]. When handling graphs, these can also be stored in compressed format [47].

Cold Storage The usage of Cold Storage, by using services like Amazon Glacier can improve how data is used in a framework, as it allows to store less frequently used data in a cost-effective manner, and allows to free up space for real-time data in more critical data storage with faster access times.

Automated Model Retraining Automated model retraining is also an interesting technique to understand, as it substitutes a cyclic training schedule for a model, on a degradation and measurements based cycle, lowering the amount of times a model suffers retraining, which also reduces the resources required by it.

Besides this, there are of course several changes and ways for models to be changed and achieve better efficiency while running the in the cloud, here are some that are deemed important to be mentioned for the purposes of this work.

Transfer Learning This consists in taking an already trained model, and retrain it for specific, more simpler tasks. This not only reduces training times, but also results in a simpler model which will grant lower prediction times, as well as boost its accuracy. One great example of this type of applications can be seen in YOLOv5. We can reduce the number of outputs to the ones desired for the model to detect and retrain it, making it a much more compact and faster model by doing so.

For instance, during my presence at FST Lisboa [9], we developed a YOLOv5 model with only 5 outputs: yellow cones, blue cones, orange cones and unknown cone. With this, the ML model got considerably faster, as we simplified to the maximum the layers of the model. Despite this, we got an increase in its accuracy, as it got better at predicting the cones in the images captured from the camera. All of this for the ultimate goal to have a model run in real time robotics in autonomous driving, so we can extrapolate similar results to ML-Ops where low latencies with still great model accuracy are also required.

[6] https://parquet.apache.org/, accessed on 10 Set 2024
[7] https://avro.apache.org/, accessed on 10 Set 2024
[8] https://parquet.apache.org/, accessed on 10 Set 2024
[9] https://www.fstlisboa.com/, accessed on 30 Aug 2024

Figure 2.3: YOLOv5 applied to a frame on Autonomous Driving.

Hyperparameter training This type of ML models training is one of the latter techniques and Bardenet et al. [48] presented a study of collaborative hyperparameter training, in which it uses surrogate-based optimization (SCoT) and ranking techniques, being essentially a sequence model based optimizer, to find the best parameters for a given model.

Model pruning Pruning is a different approach to hyperparameters, as it makes the model more compact and efficient while maintaining its accuracy, instead of finding the best possible parameters. A study made by Zhu et al. [49] proves how using model pruning can be beneficial, not only in inference times but as well as the models becoming more cost-efficient by using a large-sparse approach instead of small-dense, yielding at times even better results than the baseline models.

Knowledge Distillation As mentioned by Phuong et al. [50, 51], knowledge distillation is a technique that creates a compact ML model by using a teacher and a student model, and the student model learns the outputs of the classifier designated to be its teacher. With this the model becomes much more efficient and less complex. A Zero-Shot Knowledge Distillation framework was even developed by Nayak et al. [51] where it does not use data samples and class similarity. As expected this improved the generalization of the ML models it was used on.

Surrogate Models Along with Knowledge Distillation comes the paradigm of using surrogate models in production. This models are much simpler and faster than the original models, while maintaining the same levels of accuracy. There are many ways to produce a surrogate model, it mainly depends on the purpose of the surrogate, as well as the ML model it derivates from. It can use different methods

such as Kriging, Response surface models (RSM) or even SVMs [52]. Besides this, many surrogate models in production still have a fallback of using the original model when the surrogate is not found to be performing up to the standards set, but since the purpose of the original models is to not be used often they are usually stored in a slower access platform.

Cost-Sensitive SVM Studies have also been made regarding techniques which are solely specific to a model to investigate. Work from Iranmehr et al. [53] shows a Cost-Sensitive SVM. This model identifies that imbalanced data is a great adversity to achieve better performance with SVMs, but nonetheless the authors presented a version of this algorithm in which it enforces cost sensitivity for separable and non-separable data, enforcing larger margins. Lin et al. [54] also presented a different version of a cost-oriented SVM, named Reduced Support Vector Machine (RSVM). This one differs from the previous by centering its model in using the decomposition method, which was able to reduce training times significantly for large clusters of data, while having only a small degradation in accuracy in comparison to the standard SVM model. This makes it obvious that it is better to use RSVM for problems that involve larger datasets.

Model Ark (MArk) Zhang et al. [4] proposed MArk, an SLO-aware framework that is based on IaaS, namely AWS EC2 instances, which employs techniques such as auto-scaling and batching to make this framework cost-efficient while granting low latency to users that wish to utilise its ML services.

BATCH BATCH is a framework presented by Ali et al. [5] that shows how a framework has been devised that is able to provide to users a platform for ML services, using stateless computation (FaaS), and without becoming more expensive with increased workloads, due to using batching as an optimization technique, even performing better in comparison to MArk.

Scrooge Other applications such as Scrooge (Hu et al.) [6] are able to lower the inference time of models used online, with many mechanisms such as packing efficiency and cost-awareness, and it is able to achieve a decrease of at least 16% in costs while maintaining the latency objectives satisfied.

Clover Also exploiting the usage of graphs and partitioning workloads on GPU, Clover is a framework proposed by Li et al. [55] and it shows how clever usage of these tools can have an impact on the carbon footprint of a ML model . Another interesting contribution made to this field of study was made by Florian et al. [56] in which it was introduced predictive maintenance of ML models: a cost oriented approach based on simple inputs such as the yearly maintenance costs of false positives, false negatives and true positives of models, which is then able to reduce downtime and improve the quality of the system with simple decision making.

Finally it is necessary to mention that recently frameworks were created to facilitate the deployment of ML models such as Tensorflow [10], scikit-learn [11] and Pytorch [12]. These are helpful as both have pre-deployed versions of ML models to be used with simple library function calls, as well as by using Pandas it becomes much easier to process, manipulate and visualize data.

2.4 Summary

In the length of this chapter, it was presented many topics and fields of knowledge relevant to have a grasp of understanding for the work proposed. We covered Cloud Computation, ML models relevant to society nowadays, and to the work, as well as models in production and how they are applied in different fields of industry. Ultimately, mechanisms to handle Big Data Processing which are crucial for the well-functioning of systems in production, along with systems that are helpful for ML-Ops in production were also described, to contextualize on how various but impactful these types of work can be to the industry.

[10] https://www.tensorflow.org/, accessed on 20 Mar 2024
[11] https://scikit-learn.org/, accessed on 20 Mar 2024
[12] https://pytorch.org/, accessed on 20 Mar 2024

3

Proposed Solution

Contents

In this Chapter we present the proposed solution. We explain the methodology to analyze, explore and decide the techniques and Machine Learning (ML) models that go into the framework. We show the architecture of how the study is developed, as well as the general specifications for this implementation, such as environment used. We then formulate a Cost Model, to be able to translate the changes that the techniques make on the framework into cost, as well as algorithms to facilitate in the evaluation of the techniques.

3.1 Solution Overview

The solution consists of three tasks: a **research task**, **analysis task**, and a **testing & evaluation task**. In the **research task**, we review mostly classical ML models, and evaluate whether each model is a proper conduit of research for the goals of the study. The selection criteria is based on identical factors already mentioned in Chapter 2, where a similar preliminary research was already carried out, namely popularity of usage of a model, its applications in the relevant sectors selected for the study, and preponderance in the sectors it is used. As aforementioned in Chapter 2 the sectors taken into account are Healthcare, Transportation, Manufacturing, Agriculture and Software Development. The reason for choosing these five sectors were previously explained as well, but essentially due to relevance, and abundance of usage of ML in each sector. Essentially, the popularity of models is defined by simple characteristics, such as cross-domain success & industry adoption, interpretability & adaptability, easy-of-use, and performance and accuracy.

During the **analysis task** what is accounted is how the model behaves, and the most proper and commonly done work to make the model more compact and cost-efficient. As such, one dives into the characteristics of the model, what are its strong and weak points, its applications and evaluate how to reduce costs for this model. To make this decision it must be taken into account if the techniques or set of techniques to be used interfere with the capabilities of the model, in the sense that it makes it unable to sustain sufficient metric results to be considered an effective cost reduction.

One considerable example can be Data Pruning, although this can be advantageous for virtually any ML-Model, one needs to consider the decreases in accuracy one might have, going on a model by model case. Plenty of cluster algorithms may benefit from this, as although there can be a slight reduction in accuracy, it makes the model less complex, improving computation times. The same cannot be said about ANNs, as these are much more complex models, neural networks might work differently if not given all the data it was previously given to it before. To do this one might take into account re-training, but that would also affect the cost of applying this technique to the model.

This should always be done before running the model, either in a local or cloud environment, so as not to waste resources. One should also take into consideration the specific goals for using this model,

in a production point of view and its different usages in sectors.

Finally, in the **testing & evaluation task**, the research done is applied to a given model, as it is ran either in a cloud service (like Azure or even a simple Kubernetes cluster), or even a local cluster of machines that can simulate identical behavior of the cloud systems studied, depending on the evaluation task of the work. After running it, data is gathered to understand if the techniques found to be the best theoretical practice are actually meeting the requirements to reduce costs, without compromising the performance of the services provided. For models that showed positive outcomes, it was just a matter of documenting the results and then move to the research task of a different model, but if a model is lacking in its results we must understand why that happened and go back to the research task of said model and comprehend the changes to be made, then after that is done a new testing & evaluation task was feasable to be done.

The **analysis task** plus the **testing & evaluation task** are executed in iteration, as it is better to revolve around a model at a time until it fully meets the needed requirements, than to have the review of all the models and then proceed to evaluate them all. The reasoning for this resides in the fact that if a model needs to be reviewed again due to not meeting the goals set for it then one as a fresher memory of the matter. Figure 3.1 shows a diagram that represents the working method for the proposed solution.

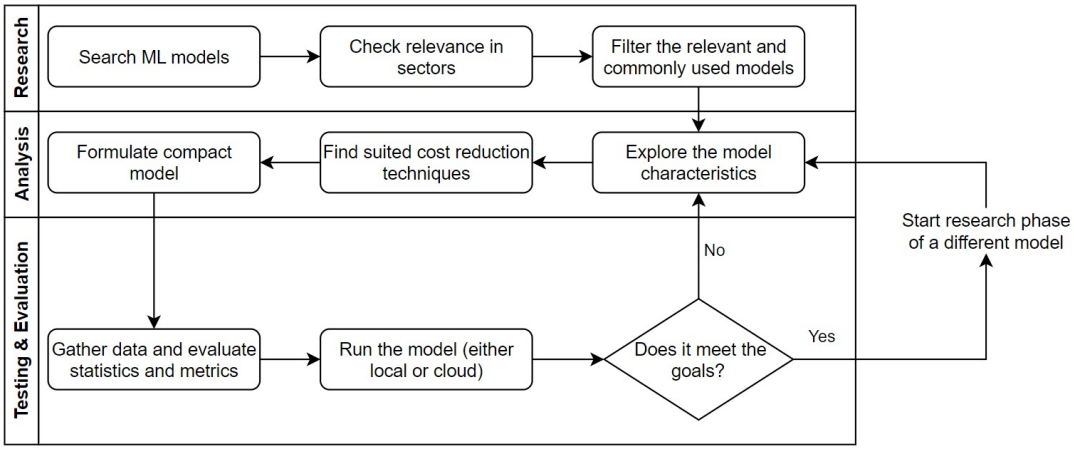

Figure 3.1: Study systematic workflow.

Table 3.1 shows what is the result of using these tasks systematically based on Figure 3.1. The reasoning for not continuing the study tasks of SVMs after the Research Task is due to the fact that other models included in the study also tackle the same purposes. Besides, it needs a significant understanding of kernel functions to be effective. Regarding LSTM also not being included in the next task, its problem comes from being a greatly complex model which translates that complexity to production. This means it could have taken plenty of time and resources trying to understand where are some characteristics of the model that when modified it leads to a more cost-efficient model. CNNs were ultimately

decided not to continue to the testing & evaluation task since the study goes over Deep NNs already, and a ML model for object detection, the need to study this specific architecture of ANNs felt uneventful.

Models	Research Task	Analysis Task	Testing & Evaluation Task
Random Forest	Sufficient	Sufficient	Sufficient
Isolation Forest	Sufficient	Sufficient	Sufficient
Deep NN	Sufficient	Sufficient	Sufficient
CNN	Sufficient	Insufficient	NA
Yolov7	Sufficient	Sufficient	Sufficient
SVM	Insufficient	NA	NA
LSTM	Insufficient	NA	NA

Table 3.1: Table of decision based on the systematic tasks.

Besides the selection made concerning ML models, cost-reduction techniques were also taken into care during this iterative study to find the better suiting for the models at hand. Although during the testing & evaluation task we focus our efforts on a single model per techniques, as Table 3.2 shows the cost-reduction techniques to utilise are flexible, in a sense that can be applied to different models than the one used to study. This grants transversality to the work, as it does not limit the results to implementation details, or specific contexts. These techniques will be explained further in the Chapter.

	Random Forests	Isolation Forests	Deep NNs	YOLOv7
Feature Reduction	X		X	X
Quantization			X	X
Parallel computations	X	X	X	X
Surrogate Models		X	X	

Table 3.2: Suitable techniques selected across the ML models chosen.

From the research made for Chapter 2 we can conclude that the models that are frequently used for production purposes nowadays are SVM [17, 18], ANNs [2, 16], and more specifically MLPs [24] and CNNs [2], clustering techniques such as Decision Trees [20] and Random Forests [21, 22]. But, for the purposes of this work we will focus on techniques related to the models that could pass on to the test & evaluation task.

With this it is natural to understand that the subsequent work to be done is to explore each model and their unique behaviors, algorithms and intricacies to evaluate the best course of action and techniques to use over the model.

It is noteworthy to mention that these iterative tasks we used to research ML model specific techniques, but during the study we are also evaluating other techniques, such as communication protocols and distributed systems.

3.2 Framework Architecture

The framework implemented is built with all the models to be explored during the length of this work, which then can be run in all testing environments, with representative workloads as well (will be mentioned in Chapter 4). For this to be possible, the work is implemented in a library, with callable methods whose calls can be made by using a REST API (using resources like Python and Docker containers).

For the cloud environment used we implement it in a basic Kubernetes cluster, to be a representative cloud environment that resembles the ones used in production, even if at a more simpler level. As far as the tools to be used to monitor, Prometheus [57] offers great monitoring services, which are also capable of storing these metrics, and then be queried when it is needed. Then Grafana [58] can be used to create dashboards, as it is a great data visualization tool.

To simplify the testing of the models we are using Docker, as it allows for the models to be containerized and be tested locally, but also Kubernetes allows for an easy transition from those containers into pods (the smallest type of instance in K8S). Besides with Kubernetes the integration of Prometheus and Grafana is trivial as well.

Regarding the metrics to be monitored, one would say that the most important techniques to analyze and evaluate are related to cost, the model performance, to computation times and latency in the response, and computation performance, which gathers data on the CPU usage and times. During Chapter 4 will be explained the reasoning for the choice of metrics.

The architecture itself consists of worker nodes which await requests to process from clients. These requests can or not be sent by a Load Balancer, whose main task is to perform simple Data Batching of the client requests, hence we are going to abstain from dynamic balancing of the environment as it is not the focus of the studies at hand.

Figure 3.2 and figure 3.3 present diagrams relative to the architecture in training and in production. Figure 3.2 displays the part of the architecture in which the *training* of the model is done. In this part, the model can be subject to two separate approaches, one in which the model suffers modifications itself, and it needs to be trained and tested again. In the other approach, there are only reduction techniques applied to the model, therefore not needing direct changes to the model, but working more like an adding layer to it. Of course this still requires testing before incorporating it to the model in the execution engine.

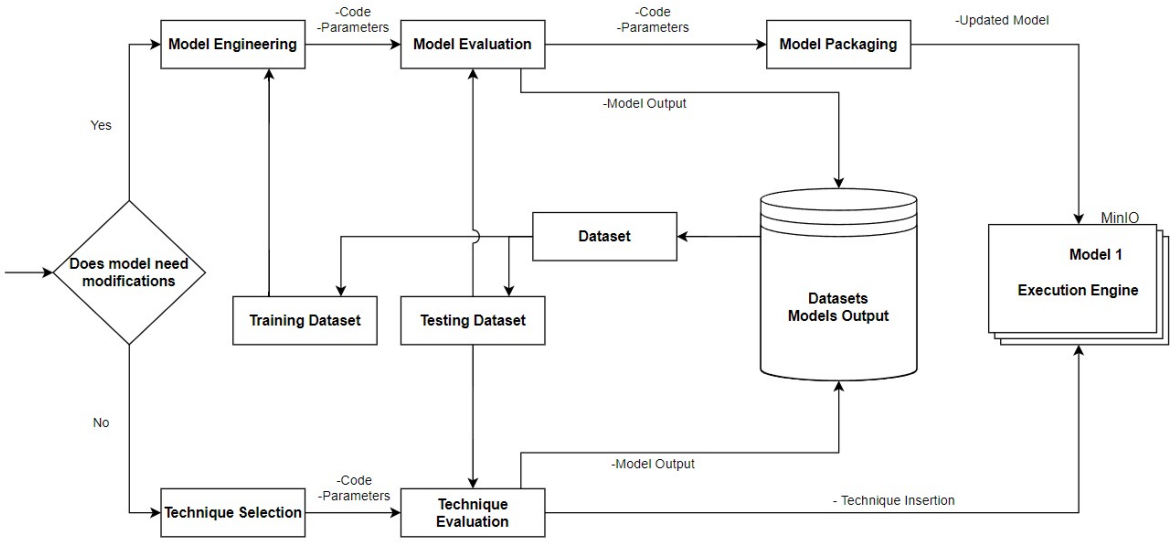

Figure 3.2: Training architecture.

Figure 3.3 is a visual representation of what the architecture of production consists of. In essence, each execution engine (all engines have the same environment when running, and can receive requests for different models) receives representative workloads to process. While all the models are running they are be monitored by the designated monitoring tool, and reproduce logs that can be stored to later be processed and analyzed. It is relevant to mention the architecture of this thesis work is based on a production ML-Ops framework called Seldon [59], as it deploys all of its services on the cloud, as well as it contains all of the data monitoring and visualization features. This framework, namely the Seldon Core one, comes with a hefty price tag, and due to this it was not considered to be used for the purposes of this work.

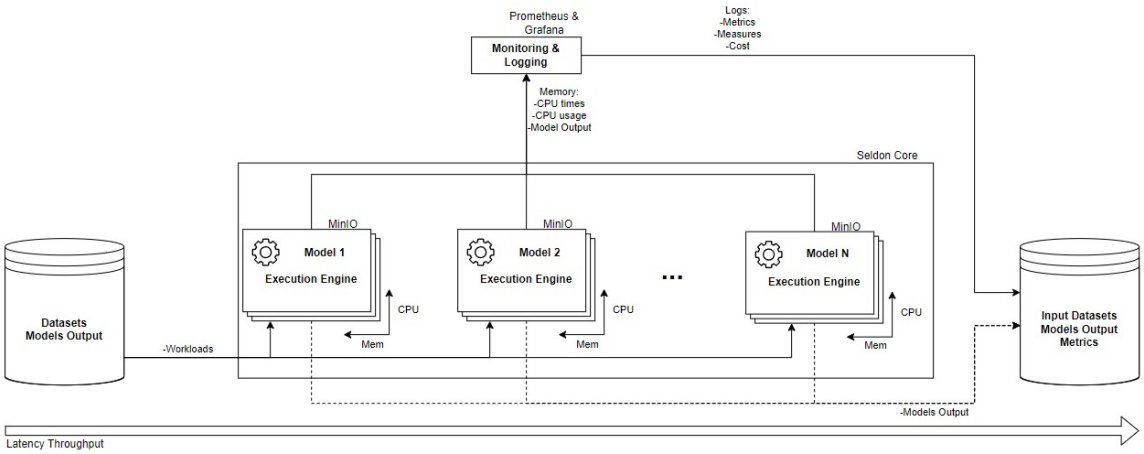

Figure 3.3: Production testing architecture.

3.2.1 Distributed Architecture

The implementation of the work follows the guidelines of what is proposed in the architecture. It is be composed of Python scripts which besides containing the ML models and the cost-efficient techniques, they will also deploy a client and worker service, to simulate a very small and controlled environment that resembles production.

All of the supporting code can be found in Appendix A, where it contains the Code of the Project.

The implementation comes in three different steps, not only to expedite the development of the framework in small and incremental steps, but to test the models in these different and easier to monitor environments before going into the final testing environment. The first step was to develop the worker nodes locally and using HTTPS requests locally. Then the next testing environment came from *containerizing* the worker nodes into Docker containers and using simple port forwarding to be able to receive requests from the client-side. Finally, came the final testing environment, which is held in a Kubernetes cluster. In it contains all the worker nodes which are each in a Pod. Then they are able to receive and send the results of the requests made by the client with a service that is established which allows it to receive requests from a local client.

3.2.1.A Framework

To better understand the framework which is intended to simulate the production environment, it will be better to start explaining how Docker and Kubernetes work.

Docker works by placing pieces of software into containers, effectively making it able to run different environments in the same machine at the same time. These containers can be replicated and used at the same since since they originate from the same Docker Image (the Dockerfile is displayed in Listing A.2). This is the basis for how the multiple Worker Nodes are deployed.

Kubernetes as the go-to engine to orchestrate containers and to test them for cloud environments felt like the trivial choice to develop the framework for this work. The architecture that is usually used in this manager is exemplified in Figure 3.4.

For the specific context of this work we are using Pods inside the same Node, not only for simplicity issues, but also due to not feeling the need to deploy a very robust environment for the purposes of this thesis (the Pod deployment YAML file is represented in Listing A.3). Moreover, during the testing of the work, there is a maximum of three working nodes listening and processing requests. The reasoning for this is not only due to computation limitations of the machines used for this, but also due to not needing an overly great number of Pods deployed to check the results pretended.

Kubernetes also contains a powerful feature which is that the Pods in a Node can utilise directly Docker Images to incorporate the application instances that are intended to run in the node [1]. This

[1] https://kubernetes.io/docs/concepts/workloads/pods/, accessed on 20 Jun 2024

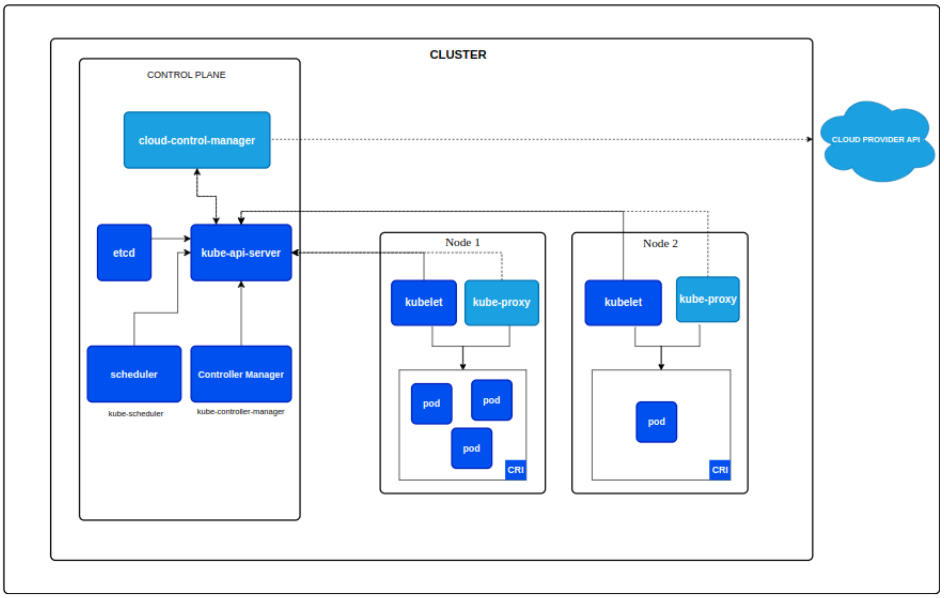

Figure 3.4: Kubernetes Architecture Overview.

facilitates the process of deploying code to the cloud environment, since the same code can be tested locally beforehand, and also implies none to not much changes are needed from one testing environment to another.

Port-forwarding rules were also created so that the network from the cloud environment is able to communicate and receive requests from the local environment, where the client-side is located with the requests to be sent (the service deployed to make this possible can be seen in Listing A.4).

Other powerful feature which can be applied to this framework due to containarizing are the already mentioned tools Prometheus and Grafana. Prometheus is a monitoring tool that retrieves important data from the framework and stores it. It also has visualization capabilites, but that is where Grafana comes into factor, as it is a visualization tool that allows the user to use Prometheus as a data source. This makes the visualization and evaluation job easier to complete and understand the behavior of the framework.

Containerizing the software to be used by using Docker containers, as well as using those images to deploy the software on Kubernetes is a great option to confer optimizations to the model serving in the framework. Not only it is an efficient solution and also enables for easy implementations of auto-scaling in the environment of the service, but the greatest takeaway of using this is that it is agnostic to software, meaning that it does not need specific software in the machine besides Docker and Kubernetes to be able to run the framework.

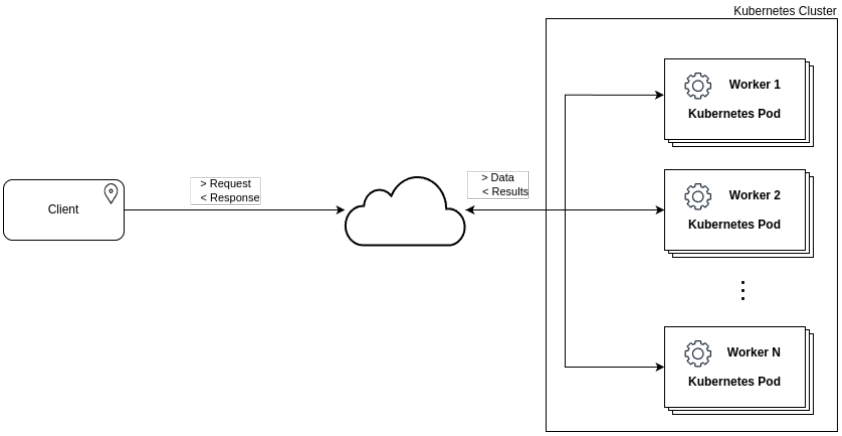

Figure 3.5: Work Implementation Overview.

3.2.1.B Communication Protocols

The main differences in implementations, asides from the ML models themselves, come from how the nodes communicate. Due to the research made it was found that exploring the communication protocol between nodes could be valuable, therefore we came to the conclusion to compare two prominent protocols: **HTTPS** and **gRPC**. The reasoning for choosing these two is based on being currently the most famous and used request/response protocols used worldwide.

HTTPS [60] is an extension of the HTTP protocol, which ensures better security to both parties that are establishing communication from attacks. It ensures confidentiality, integrity and data authentication [61]. Since this is one of the most used protocols for the past decades, it was decided to be the default protocol for communication in the framework.

gRPC [62] is a Remote Protocol Calls (RPC) service developed by Google which is built using HTTP/2 but contains some key features which distinguishes it from HTTPS, the main one being the usage of Protocol Buffer. Unlike HTTPS which uses JSON to serialize its data, gRPC uses Protocol Buffer which is a mechanism that allows the used to serialize data in a structured manner [2]. It is done by creating *.proto* files, which are essentially the files that set the rules for the requests and responses to be communicated. With this, we are free to create different requests with their corresponding responses that contain customizable messages which must also be specified in the *.proto* file.

Then came the simple task to for each implementation create the JSON data structures for each request needed to be done on the server-side, as well as defining the data structures to handle gRPC communication in the *.proto* file, which is shown in Listing A.1. As mentioned, the JSON data structures follow very similar structure to what is display in this ProtoBuf contract.

[2] https://grpc.io/docs/what-is-grpc/core-concepts/, accessed on 20 Jul 2024

3.3 Techniques Integrated in the Framework Pipeline

The general work of this thesis is to integrate cost reduction techniques on the three basic modules of which the pipeline of the framework consists of, which is represented by the flowchart displayed on Figure 3.6, hence the focus of the work is mainly around each of those three aspects of ML-Ops: Data, Model and Inference. The three of those are crucial not only for ML related aspects of performance, but also at a production point of view, there are valuable changes that can be made to them to achieve the desired results.

The figure shows each module connected to each other in a sequence, as one change in a certain aspect, can have impact on another later. What it means is that for instance any change made by a technique applied to data, can also have repercussions for example in the model or even the inference of the framework. We delve deeper in this in the following subsection of Decision Criteria.

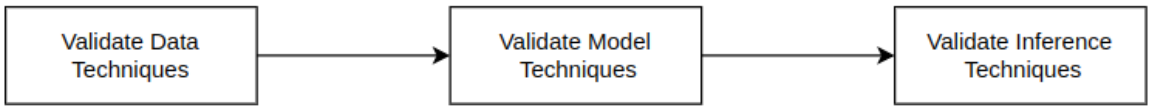

Figure 3.6: General flow of technique validation.

With this division settled, then came the task to name the most suited techniques to apply to each of these three aspects of the framework. Ultimately there were select two techniques for Data Related, four for the Model Related, and three for the Inference Related, shown in Table 3.3.

Data Techniques	Model Techniques	Inference Techniques
Data Pruning and Transformation Data Optimization	Feature Importance/Selection - Clustering Parallel Computation Quantization - Deep NN Quantization of YOLOv7	Communication Protocol - gRPC Batching Requests Surrogate Models

Table 3.3: Cost Reduction Techniques Studied.

These techniques were derived by taking into account the most prominent models used in all sectors, based on the knowledge and research gathered for Chapter 2. Clustering ML models for example sees a lot of usage in Healthcare, as well as time-series anomaly detection, useful for Manufacturing for instance.

Deep NNs sees a lot of usage across all of the industry, either it is Agriculture, Transportation or Manufacturing, so it would be very complex to go over every use-case due to its flexibility to be applied to plenty of situations. For this reason, we will be doing a more generic approach, but insightful and horizontal towards how these models are used nonetheless. Finally, we will go over a specific case of a Deep NN, YOLOv7, which it started of with a CNN architecture in its previous versions, as it is

a benchmark for object detection in the recurring days, hence also important in fields like Healthcare, Agriculture and Transportation.

3.3.1 Techniques Overview

Now we go over each technique adopted for the purposes of the study. These were selected on many factors, such as complexity of employment in production environments, performance and benefits leverage, as well as the ML models they are centered around. The techniques come second to the models elected since these should be applied in production hence the need to select a relevant few, factor which is measured by the popularity and flexibility of usage of said model across several sectors. Not only this but some of the techniques studied do not come from model or inference based roots, rather they are ML-Ops oriented technique that do not need to be studied case by case for each model, but can be combined with model oriented techniques to achieve better results.

3.3.1.A Data Techniques - Data Pruning, Transformation & Optimization

Data Preparation comes as a field that is not solely correlated to ML but also to other related fields such as Data Mining or Data Integration. This process allows for better efficiency on the task for it to be used on, either it would be for the first steps of a KDD, or a ML model.

The importance this process brings to this work is in the sense of models that raw data can have a tow in its performance. It consists in cleansing the dataset used in many aspects as it can delete incomplete, inconsistent or noisy data entries, as well as creating a smaller dataset with better quality [63]. This comes from iterative steps of data quality checks, cleaning, and transformation and integration as well [64]. Although this proves more advantageous to Data Mining processes, this also benefits the performance of ML models.

The main techniques that are studied are **Data Pruning, Transformation and Optimization**. Data Pruning consists of making the dataset that is used on models simpler, by removing variables, making the data less heavier, and simpler for the model to process. This is high correlated to the Feature Importance which is overviewed as well. Data Transformation is responsible to apply changes to the dataset to make it enhance its usability. Actions like cleaning or restructuring the data can be seeing as parts of the process of Data Transformation. Data Optimization focuses on different matters: it ensures that the storage is as efficient as possible, the data queries and analysis are quick to be fulfilled, all the while maintaining efficiency patterns.

3.3.1.B Model Techniques

Feature Importance/Feature Selection Feature Importance is a great method to obtain information from models which are tree ensembles in which dimensions from the input data are the most relevant for the model to make a prediction. This comes in useful when one wants to derive a more compact model from a clustering model [65].

This is where Feature Selection comes into the equation as a way to create smaller, faster models. It takes into account techniques similarly to Feature Importance and then removes the *n* less significant features of the input data in order to achieve better efficiency while not taking a toll on performance. The technique chosen for this study is a tree based feature importance algorithm due to this technique being used mainly on random forests [65].

SHAP-technique Feature Reduction Shapley Additive Explanations (SHAP) is a game theory based technique that has been proved to be plenty useful to detect the importance of features in a model prediction. It is based on game-theory, using the prediction as the payout, and the players being the features: each receiving more income by the importance of their role in the prediction [66].

This is a notoriously robust algorithm to apply to clustering algorithms when compared to others such as LIME, since it is faster computationally and both are model-agnostic, but offers insights more correlated to specific outputs and general outputs of a model from a certain dataset [66].

Then to apply it is much like a feature reduction technique in which it is evaluated the output of the SHAP values for each feature and the *n* less relevant features are not further considered to be used in the clustering model. Much like the method explained before, it will have an impact in computation times and accuracy of the model.

Parallel Computation Parallel Computation is a different paradigm in programming which allows for programs to stop being run sequentially but rather in parallel, by means of doing multiple computations at the same time. This can be useful when there are loads of operations needed to be done and there are bottlenecks that can be eliminated by running different modules of a program in parallel.

Quantization Quantization comes as a technique that is more directed to Supervised Learning models, specifically ANNs. As these models produce weights as output of their training, which are parameters that are saved and will be used in the future whenever it is needed to make a prediction with said model.

Quantization comes into play by taking into account the complexity of the weights. Usually these weights are constituted of floating point numbers which are quite large bit-wise and will make it harder in computations time-wise as well. By quantizing those numbers to integers for example, or even floating

points that use up less bits not only the weights will take up less space, but it will make computations a lot faster. Quantization can be applied while training the model, or post-training, but the focus of this study is on post-training quantization.

There will be two types of quantization: static and dynamic quantization. Both of them have the weights quantized, the crucial difference comes in the activation. Static quantization has the activation of the layers already quantized whereas dynamic quantization has the activation of the layers done while inference is occurring. This comes with their up and downs, which will be breakdown during its evaluation.

Quantization of YOLOv7 Not unknown to many, YOLO is an object detection model that has evolved much in the past few years due to its capabilities, which made many researchers in the community contribute to this model and create different versions. Although its backbone consists of a classic CNN architecture, to meet higher standards and more demanding requirements this model suffered plenty of changes along the years, from switching its framework from DarkNet to Pytorch, to reducing the number of parameters [67]. YOLOv7 comes as a cemented version, well-received by the community which lead to its vast use across many applications in Computer Vision, hence this study focus on making this model more efficient as well [68].

Since this is a very robust and complex model, with a well-established architecture, the best way to try and tackle this issue is to quantize its weights. Due to the complex nature of the NN one must try different approaches to quantization to understand what is the best way to quantize the weights to ensure a more efficient version of YOLO, as well as effective.

3.3.1.C Inference Techniques

Communication Protocol Another technique that is not entirely correlated to ML, regarding this work it consists in comparing between the HTTPS and the gRPC protocols. The goal for this is to mainly compare request and response communication times. This is a reasonable matter to be studied due to the rise in the need for ever faster latency times in requests made to services in every technological aspect.

Data Batching This form of computation does not come from a ML root but rather from Dev Ops. This technique essentially employs the usage of more than one worker node at the same time for one single request. It comes from the assumption that by giving a model a smaller input it will produce an output in a smaller computation time. Although this sounds linear at first, one must take into account other aspects that is shown in the evaluation of the work, for example the overhead of already having three worker nodes up or having to create them solely for the purpose of one request.

Surrogate Models From what was mentioned in Chapter 2, this technique is derived by using Knowledge Distillation, and the outcome of this is what it is called a Surrogate Model. This simplified version of the original model is made of a simple binary decision tree that studied the outputs of the more complex and robust first model. It ensures that the results are not much degraded in comparison to the teacher model, while obtain much faster computation times.

Surrogate Models themselves can be seen as simplified versions of ML models, but when it comes to ML-Ops there are many preoccupations that should be taken into care. These models are not nearly as complex as the models they substitute, so one should be secure that they are viable for production. The model that was evaluated is an Isolation Forest that is used for anomaly detection in a time- series. The surrogate model consists of a Kriging Partial Least Squares (KPLS) regression. It combines the Kriging Gaussian Process with Partial Least Squares.

3.4 Cost Modelling

In this section we delve into deeper understandings of what cost is for the goals of this work. This is important to understand as it is one of the main metrics to take into account, and although it seems a simple domain, it can become vague when plenty of variables that are cost-derivative must be considerate.

For this, we strive to englobe all of the aspects that are relevant from the standpoint of companies regarding the optimization of ML models. To realise this, we formulate cost formulas for every dimension of cost that it is justified to exist for the purposes of evaluation.

3.4.1 Domains of Cost

All of the types of cost that are mentioned are not straightforward variables, as they are related to the aspect of the framework which they affect, so it felt important to firstly describe the primitive domains which leverage the changes in the cost variables that will be described ahead.

Space Space can be a very simple dimension to evaluate, but nonetheless still very important to take into account since it is one of the main aspects to look for billing factors in cloud environments. It is important to understand that there can be changes made to make models smaller, but also there are techniques that result in a decrease in size, but have no direct changes in the model that require the models to be trained again.

This is the case with quantization, which changes parameters and weights to go from structures such as floats and doubles to integers which require fewer bits for example. But there are times where changes to the model itself are made in order for them to become more compact, and one of the most notorious changes is the space it occupies, as it suffers noticeable decreases. The obvious and simplest

way is to directly compare the space usage of models, using a ratio between the sizes of the model before and after the technique can also help compare.

Time Time can be seen as a technique of cost that works similarly to **Space**, as the billing of VM containers also takes into account the time used, but is has a different aspect related to it. Time is a highly correlated with the computing capabilities of the VMs where the models are being run, hence time of computing is limited by the hardware. Despite this, it is fair to say that this is still a relevant technique of cost and the most direct way to evaluate is by using a ratio of computing times between pre and post application of cost reduction techniques.

ML Metrics On another note, ML Metrics are also another insightful way to understand how the models are behaving after being applied cost reduction techniques, so it is very fitting to include them as a technique of cost. These can vary from model to model, as different measures such as F1 score, accuracy, or the Receiver Operating Characteristic (ROC) curve evaluate different aspects of a model. Therefore, ML metrics are an important source of information to justify when some techniques will result in an increase in cost, as well as explaining some increases in cost, to be in compliance with the SLOs defined.

Hardware Hardware is a central aspect of both Cloud Computing and Machine Learning, but for different reasons. In Cloud Computing, different provisions are offered to clients, which can vary on the machine specifications, namely GPU, amount of memory, CPU power, disk memory. Overall the impact that this makes is in the performance of the cloud environment, as well as the monetary costs related to using this environment. Regarding Machine Learning, the hardware used can also be impactful, as with better hardware, and using hardware acceleration techniques gaining advantage of GPU usage, can grant the models better performance, not only regarding ML metrics, but computing times as well.

The main issue is that hardware is limited by the budget and company choices for the specifications of the VM containers that are used, and those can also directly impact the performance of the models. Some models can use techniques that imply great usage of GPU and the container is not equipped with a well suited unit for example. Ultimately the decision for what hardware is used is based on the choice of balance between performance and costs of the provisions.

3.4.2 Types of Cost

Now that the basic foundations of cost have been explained, we delve into what are the key factors to take into account while evaluating the behavior of the technique applied in the framework to make the ML models more cost-efficient to be provided in the cloud.

By employing an analogous process to the technique, we discriminate cost into three different types: **Data**, **Model** and **Inference**. Figure 3.7 shows the scope of each type of cost, which will be presented afterwards.

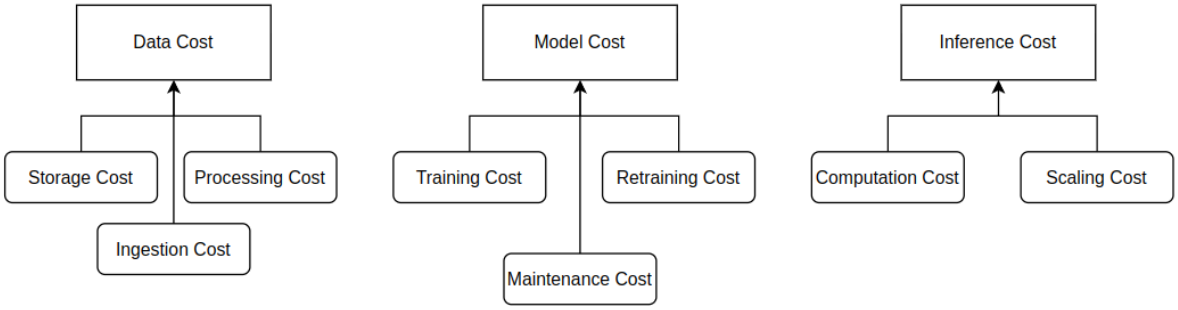

Figure 3.7: Types and scope of Cost.

Data-Related Costs These are costs associated with acquiring, processing, and managing the data required for training, testing, and inference. They are horizontal to the whole framework as data is a central piece throughout the whole pipeline, and take different aspects into account in different parts.

The key aspects for this are **Data Storage**, **Data Processing** and **Data Ingestion**.

Data Storage is focused on how the data is stored and its storage solutions for the whole framework. **Data Processing** on the other hand takes into account all the changes that are done to the data, either it is cleaning, transforming or preparing the data. **Data Ingestion** comes as the criteria for analysing the streaming of data around the network.

Model-Related Costs Model-Related Costs are more focused on the expenses that it takes to modify the model itself, and then the subsequent preparation for the model to be able to be used. This means that it is focused on its **Model Training**, which englobes both the initial training, and necessary subsequent retraining, and also **Model Maintenance**, more focused on version control, testing and debugging of the model.

Inference-Related Costs The focus of this type of costs is centered around the production environment itself, and how the models behave in said environment.

It includes **Compute Costs**, which take into account resources required to make the requests, model complexity and latency requirements. It also includes **Scaling & Deployment** which as into its scope matters like service scaling, batching and fluctuating demands.

3.4.3 Formulas

The main reason why we discriminated all of these type of costs was to be able to also have a way to quantify the techniques to apply to ML-Ops, making it an easier task to evaluate the work. Therefore all types of costs will turn into formulas which have its importance on the global evaluation of each technique.

Data Cost (C_{data})

$$C_{data} = C_{storage} + C_{processing} + C_{ingestion} \tag{3.1}$$

Where $C_{storage}$, $C_{processing}$ and $C_{ingestion}$ are the costs related to data management.

Cost Model (C_{model})

$$C_{model} = C_{training} + C_{retraining} + C_{maintenance} \tag{3.2}$$

Where $C_{training}$, $C_{retraining}$ and $C_{maintenance}$ are the costs related to model training and maintenance.

Inference Cost $(C_{inference})$

$$C_{inference} = C_{compute} \times N_{requests} + C_{scaling} \tag{3.3}$$

Where $C_{compute}$ is the cost per inference request and $N_{request}$ is the number of requests. $C_{scaling}$ includes the auto scaling and infrastructure costs.

Total Cost (T_{cost})

With this, we can formulate the final cost formula as the sum of the previous proposed cost formulas:

$$T_{cost} = C_{data} + C_{model} + C_{inference} \tag{3.4}$$

3.5 Decision Criteria

Although all individual aspects of the work have been covered and explained, the decision criteria is still an important matter to be explained, since it is due to its nature that we shall ultimately decide whether the new technique implemented is effective at its goal, or it should be discouraged of usage.

There are essentially two different domains that are important for the decision-making process: **Cost** and **Performance**. Cost has been already described in the previous Section, hence there is little need to justify its importance for the decision process. Performance on the other hand, is a very intriguing matter to take into consideration. It should be mainly separated into two types: **ML related** performance and **hardware/computing** performance.

ML related performance comes from the metrics that are extracted out of each specific model. Hence this type of performance is correlated to the outputs of accuracy, ROC curve, and F1 score, to name a few. This is important to take into account, not because of efficiency techniques, but due to Service Level Objectives (SLO) beforehand established upon between the service provider and its contractors, which are made sure to be maintained by creating Service Level Agreements. One needs to make sure that by applying the technique at hand, there will not be a severe decrease in performance which violates the performance agreed in the SLAs. This is done by defining a baseline in ML metrics that should not be lowered at any given point while the service is deploying in production.

Computing performance is justified by the same derivatives as ML metrics. The *speed* of how a response is given by the service to the client is not only an issue cost-wise as the more time it is spent computing the more expensive providing the service becomes, there are SLAs which establish a minimum time-to-respond as a baseline which should never be violated.

3.5.1 Decision Algorithms

With this settled comes the task to define how to evaluate a technique. For this, it was chosen to base these decisions on the Cost Model that was defined. This means that any change made in the framework will have an impact in at least one of the following: C_{data}, C_{model} and $C_{inference}$. If that is not the case, then it must be justified by ML metrics, since its the other type of metric that exist that can show behavioural changes in the framework.

Cost of Data Changes regarding cost of Data are straightforward since it has a go-to rule: whenever there are changes applied to data itself, whether it is related to storage, processing or ingestion, it will naturally result in a decrease in cost. The reasoning for this is that any technique that directly applies changes only to the cost of data strive to reduce costs related to data.

Therefore we can safely assume that all of the changes and evaluation is done based on what Algorithm 3.1 shows, and that $costChanges$ will be normally be a negative value. Although one needs to take into account that some techniques whose focus is not data, can have an impact, and at times a negative one, but that will be covered further in this Section.

Of course, one should also assert the impact those changes bring regarding metrics and evaluate how beneficial those are for the framework workflow overall.

Algorithm 3.1: Cost of Data decision process

```
begin
    if changes applied to Data then
        C_data ⟵ C_data + costChanges
        assertModelCost()
```

Cost of Model On the other hand, whenever changes are applied to the model specifically, it will result in an increase in cost. This happens due to model covering the costs for the training, re-training and maintenance of models. The way the decision is made regarding cost of model changes can be seen in Algorithm 3.2.

Algorithm 3.2: Cost of Model decision process

```
begin
    if changes applied to Model then
        C_model ⟵ C_model + costChanges
        hasInferenceCostIncreased ⟵ assertInferenceCost()
        if hasInferenceCostIncreased === True then
            assertMLMetrics()
```

A key and interesting aspect of this process is that it checks if there were any changes made regarding inference costs. The reasoning for this comes from the fact that the changes made to the model usually have an impact in the serving of the model in production, namely the computation costs. Thus, one needs to check if this is the case, and especially if it actually reduces the cost of inference like intended. If that is the case, then if the technique is properly implemented, over time the reduction in costs of inference will outweigh the costs of model necessary to make the change. Else, one needs to check the ML metrics extracted from running the model after being applied the technique to analyse the behaviour of the model.

This can lead to a few outcomes, for example, one can find that although the technique lead to an increase in cost, the model is now operating as expected, and we can give a final evaluation on the matter. If the changes are not up to par to the expectations, then it is issued a new analysis task for the technique (and perhaps model) at hand.

Cost of Inference The processing of the computations of the cost of inference can be seen in Algorithm 3.3. When it comes to evaluate the changes made cost-wise in inference this comes as a two-part process. In the first *if* case, we check for changes in the cost of scaling of the system in the framework. This is fairly simple since it only affects one variable on the formula. Then the next step is to check for changes in the number of requests, as well as the computation costs in inference of the model, which is represented by the second *if*.

Unlike cost of scaling the cost of computation and number of requests are not independent. They are correlated, as there can be techniques that will reduce the computation costs but increase the number of requests made to the framework, or vice-versa. For this to be properly analysed, we compute the product of the cost of computation with the number of requests if there is any change made to either of these variables.

Then, the new cost of inference is computed by adding the product and the cost of scaling. Finally, comes what is shown in the last if then else case, where it is asserted whether the new cost is effectively lower than the original cost, and then comes the simple decision that if it is lower we accept the technique as valid for the framework. Else wise it is discarded on its current state, and it either is fully discarded, or even it goes back to the analysis task to suffer new changes, so it can be tested again and checked if it now serves its purpose and goals.

Algorithm 3.3: Cost of Inference Decision Process

```
begin
    if changes applied to Inference then
        if scalingChanges === True then
            C_scaling ⟵ oldC_scaling + costOfScalingChanges
        if numRequestsChanges === True or computationCostChanges === True then
            computationRequestsProduct ⟵ N_requests * C_computation
        C_cost ⟵ costComputationRequests + C_scaling
```

Total Cost One key characteristic about applying cost reduction techniques is that there are times when applying a technique with the intention to reduce a certain aspect of cost, there can also be impacts on different aspects. What this means is that one should be careful when applying techniques and leverage whether the technique in the overall cost total is not increasing the cost, rather than lowering it.

Algorithm 3.4 shows what needs to be done to take the overall cost changes into consideration.

Algorithm 3.4: Total Cost Decision Process

```
begin
    if changes made to any Cost Module then
        C_module ⟵ updateCost()
    C_total ⟵ C_data + C_model + C_inference
    if C_total < oldC_total then
        accepttechnique
    else
        discardtechnique
```

3.6 Summary

In this Chapter we described what is the Solution Proposed to achieve the results expected from this work. First we described a simple workflow to how we would research, explore and evaluate techniques to implement in the framework. After this we described the architecture on which this framework would be built and tested upon. Then, the techniques that were selected for this study were presented and given a brief explanation how they work for the purposes of this work. It was also formulated a cost model to be able to better evaluate the impact of the techniques on the framework. Finally, some implementation details were shown, which are necessary to be explained to comprehend how the framework operates.

4

Evaluation

Contents

During the length of this Chapter is where we go over the results of the solution proposed in the previous Chapter. Specifically, we go over not only the metrics obtained themselves, but also correlate what was obtained with the objective of the work, hence we translate the metrics into the definitions of Cost proposed earlier to incorporate into the Cost Model. Furthermore, some specific implementation details are displayed to help understand the results obtained.

4.1 Context

Real workloads can become quite expensive with usage over the time, and for companies this problem only becomes aggravating. One realistic example of how much a company spends in a month is presented in Table 4.1. This is a cluster from Microsoft Azure using a VM with the specs from a NCasT4_v3-series [69]. As it can be seen by analyzing Table 4.1, this comes at a monthly cost of roughly $383.98, and the respective VM is only handling a medium-low volume of requests for YOLOv5 requests. Since the payment method of this service is ”pay as you go”, one must carefully manage how its available resources are being spent, therefore the necessity to reduce the costs.

Another way to test the results of this work can be a direct comparison to cloud deployment services. ML services can be offered this way to clients by different cloud providers, one being Amazon Sagemaker [3], but they can still be costly, therefore one way to evaluate if our work is relevant in terms of cost reduction is to check if the results are better than cloud deployment, not only in cost, but also in latency.

Table 4.1: Example of realistic VM specifications.

Example Cluster Specifications	
Region	US-East
Service Type	Virtual Machine
Series	Standard_NC16as_T4_v3
Service	Pay as you go
Cost Estimate	$383.98

With this, it is clear that the objective set to reduce costs of ML models is not irrelevant matter, even more without creating complex models that take too long to compute, and it is not too much far fetched to have workloads and cost estimates like the one presented to serve as a baseline for improvement.

Furthermore, defining Cost as it was done becomes important not only since all of those aspects discriminated can translate into money spent daily on cloud services by ML providers, as this provides a more clear and palpable variable to understand, but also from the already mentioned aspect of maintaining the service up to bar to what was established by the SLOs, even after the changes made.

4.2 Evaluation Work

The testing done was conducted in different and gradual environments, not only to understand how the changes made to the framework behaved, but also to allow for a swift and easier implementation of code and structures.

We go over each of the techniques researched in the previous Chapter, and explain how the results translate from what was the theoretical basis of this work.

4.2.1 Data Preparation - Data Pruning, Transformation & Optimization

All of the alterations that are done in this technique will solely affect the cost of Data in the framework. In this we concern with the changes that were made regarding the data storage and transformations that were made that were not mandatory for the models to function. This means applying things data loaders to the models, or data pruning and cleaning of the data.

Data Pruning Data Pruning is one of the most prominent concepts used during the study, especially for Feature Reduction, which reduces features based on a given heuristic. The focus in this subsection will be specifically towards the impact this technique has on data, as for the rest it will be explained further ahead.

Relating to the implementations of the Feature Importance/Selection techniques, we have two similar in theory hypothesis: a Random Feature Selector and a SHAP-based. The key difference between these two for the purposes of this technique specifically is that in the Random Feature Selector the amount of features that remain after pruning is seven, and in the SHAP-based Feature Reduction is ten.

The difference it makes in the data aspect is essentially made to the dataset size. The original dataset had thirty two features, so the reduction in features for both alternatives is quite significant, which will have a noticeable impact in the cost of data.

The original dataset file size is of 125.2kB, and by pruning it, using the Feature Random Selector we obtain a dataset with 15.2kB and using SHAP we end up with a dataset of size 22.8kB, so a percentual decrease of 87.86% and 81.79%, respectively. This might not mean much storage-wise, so $C_{storage}$ will not suffer a notorious decrease, but still a welcoming one. The aspect of data which is affected the most is the data ingestion, since by simplifying the dataset, it also facilitates its ingestion, which ultimately reduction $C_{ingestion}$.

All of these matters described just add up to conclude that Data Pruning plenty of help to make ML-Ops more cost-efficient, so it should be present in most of the systems that are ML related.

Data Transformation Data Transformation is the task to transform data from one format or structure to another. The principle here is to make it more suitable to use throughout the framework.

Throughout the framework there are many actions done to transform data, such as normalization and data encoding. The principal actions are the label encoding done to categorical features, which are transformed to normalized numerical features, as well as regarding the images used as input for the MLP used to test Deep NN techniques. In this, the data is originally in images, which are transformed to pixels to make it possible to send, and then received by the worker in pixel values, which transform that data back to arrays of images, and subsequently transformed to a data loader.

These transformations overall allow for an increase to the cost of processing of data ($C_{processing}$), which might seem odd, but the purpose of this transformations is to facilitate the computations done by the models, and the framework itself, which will be evaluated in the following techniques described, which apply these data transformation techniques.

It can be stated that $C_{ingestion}$ would suffer a decrease, but since images are not the natural dataset for all ML models tested in the framework this would not be applicable, as well as this decrease being pretty negligible by terms of cost itself.

Overall, data transformation can be seen as valuable not by inferring cost drops in the framework, but as a technique which enables better performance and results to other techniques in this framework.

Data Optimization The only and principal technique applied to the framework regarding Data Optimization was data batching. This technique is applied in the context of using MLPs in the framework, in which its dataset is based on images. Since this ML model deals with complex data having batching incorporated into the framework. Essentially, the data loader data structure which is used can be used for batching as well, which enables for better model performance.

Figure 4.1 shows a plot which contains the computation times of the default MLP model used in the framework, and then the same default MLP model but using batching.

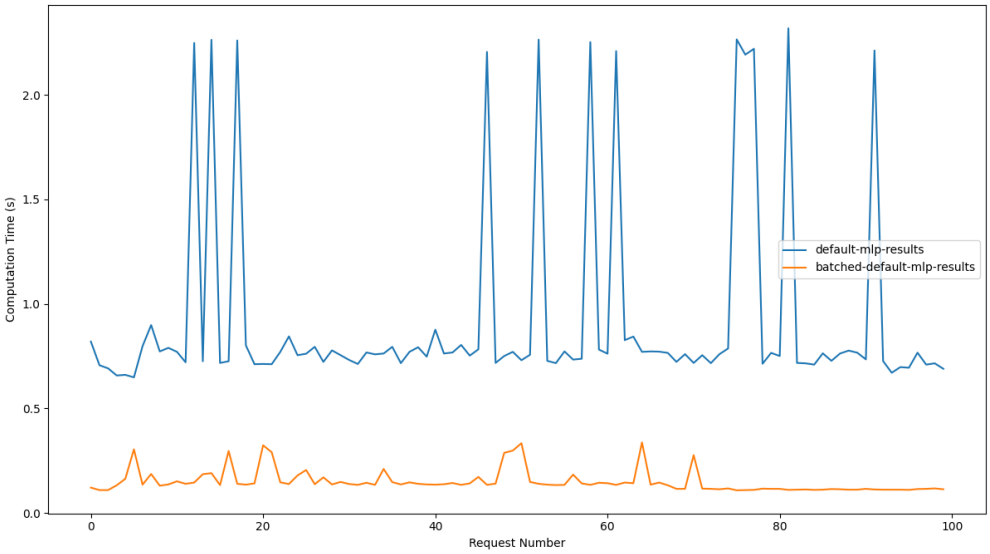

Figure 4.1: Comparison of computation times of MLP model with batching.

As we can see utilising batching of data is much faster than simply not, achieving an average difference in times of 0.7792 seconds (a decrease of 70.84%). This shows premise regarding cost reductions in data. It is trivial to understand that the major impact comes from a decrease in $C_{ingestion}$, which will lead to a reduction in C_{cost}.

In a final remark, utilising techniques that manipulate different aspects of how data is handled in a framework grant many benefits to a framework. Table 4.2 sums up the pros and cons of utilising these techniques.

Pros	**Cons**
Data size reduction	Small computation overhead needed
Faster model processing	

Table 4.2: Pros and Cons of using data techniques.

4.2.2 Feature Importance/Selection - Clustering

As aforementioned in Chapter 4, there were two types of Feature Selection studied for this framework: a regular Feature Selector and then SHAP based Feature Selection. The first one is based on a simple Feature Selector, in which its behavior is shown in Algorithm 4.1. Then the seven most important features (based on their scores from the algorithm) were kept, and the resulting dataset given as input to the Random Forest dataset.

Algorithm 4.1: Feature Selector Code Breakdown

begin

$numIterations \longleftarrow baseIterations + multiplier * log(numFeatures)$

for $i \longleftarrow 1$ **to** $numIterations$ **do**

$randomLabels \longleftarrow randomfloat(0, 2)$

$model \longleftarrow RandomForest(params)$

$model.fit()$

$importances \longleftarrow importances + model.featureImportances$

Figure 4.2 shows the Receiver Operating Characteristic (ROC) curve of what we will be calling the default Random Forest model. It is a classic and simple model with no alterations made to it, and it serves as the baseline for not only this technique, but also others that will be evaluated ahead, therefore this figure is important for the studies to follow as well.

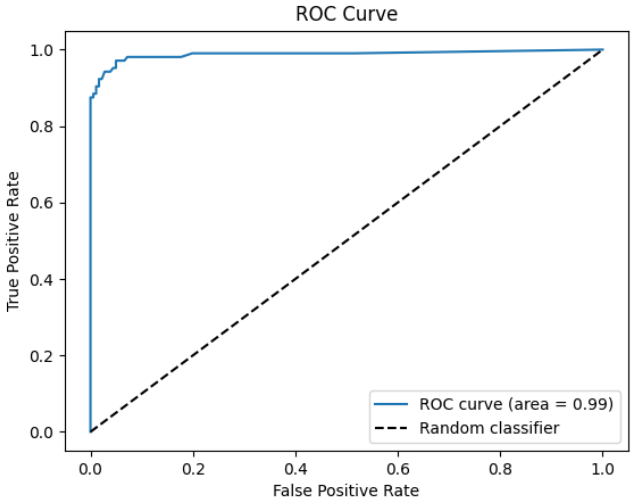

Figure 4.2: ROC curve of a classic Random Forest model.

With this said, the ROC curve it produces is the one described in Figure 4.3. As we can see the Random Feature Selector produces an Area Under the ROC (AUROC) curve significantly smaller than the original model. This can be explained by the Random Feature Selector not having an optimal heuristic and at frequent times removes variables that are impactful for better prediction, thus accuracy.

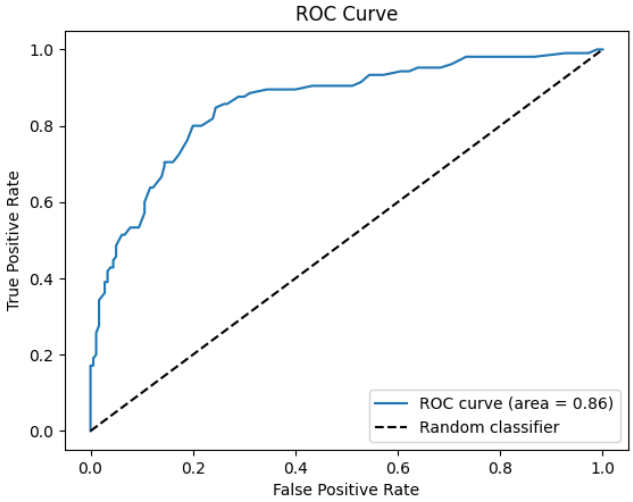

Figure 4.3: ROC-Curve of Feature Selector.

The SHAP-based Feature Selection takes a different but similar approach to obtain the same results. The procedure is essentially the same, but instead we used SHAP-measures to compute them. With this said the results can be seen in Figure 4.4.

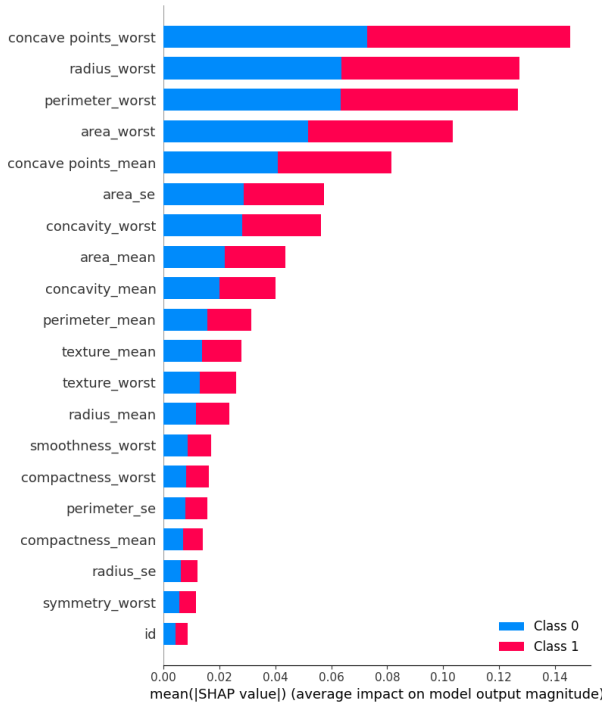

Figure 4.4: Results of SHAP-based Feature Importance.

Then, the next step was to filter out the less relevant features by order out of the dataset. As men-

tioned before, we ultimately rendered out features until we were left with 10 out of the 32 features in the dataset to give as input to the Random Forest model. The decision came empirically, by trial & error. The reason for this model having more features than the Random Feature Selector is due to the computations needed to gather 10 features would be a lot longer than SHAP, and when testing Random Feature Selector, adding three more features would not induce much differences in terms of performance.

With the final amount of features decided, this was the ROC curve that was produced from that final model, pictured in Figure 4.5. This ROC curve one looking much close to the original models when compared to the one produced by the Random Feature Selector, which indicates that its prediction are also closer to the original mode.

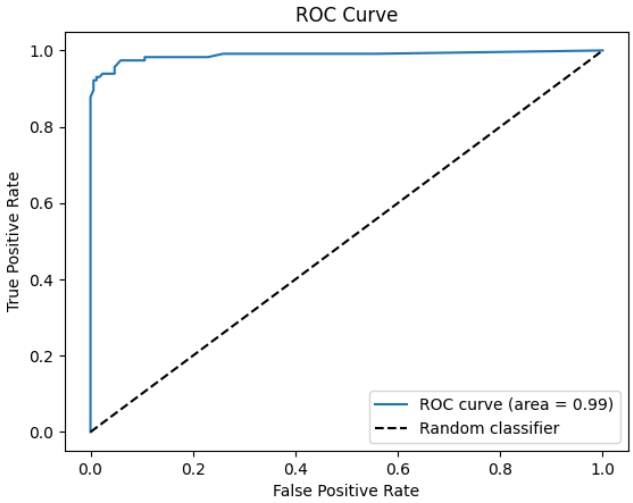

Figure 4.5: ROC-Curve of SHAP-based Feature Reduction.

It is also important to take into consideration computation times, as making the dataset simpler can also produce changes in computation times of ML models. With this, Figure 4.6 shows a comparison between the three types of Random Forests evaluated in this technique: default, basic random feature selector, and SHAP-based feature selection.

In terms of computation times we can see just by looking at the graph that the default Random Forest is the slowest, the Random Feature Selector is slightly faster than the default Random Forest, and the SHAP-based Feature Reduction is the fastest of the three. This is backed up by the average of the computation times, which shows that the Random Feature Selector is on average 0.00279 seconds (11.16%) faster than the default model, and the SHAP-based is 0.00527 seconds (21.08%) faster than the default model.

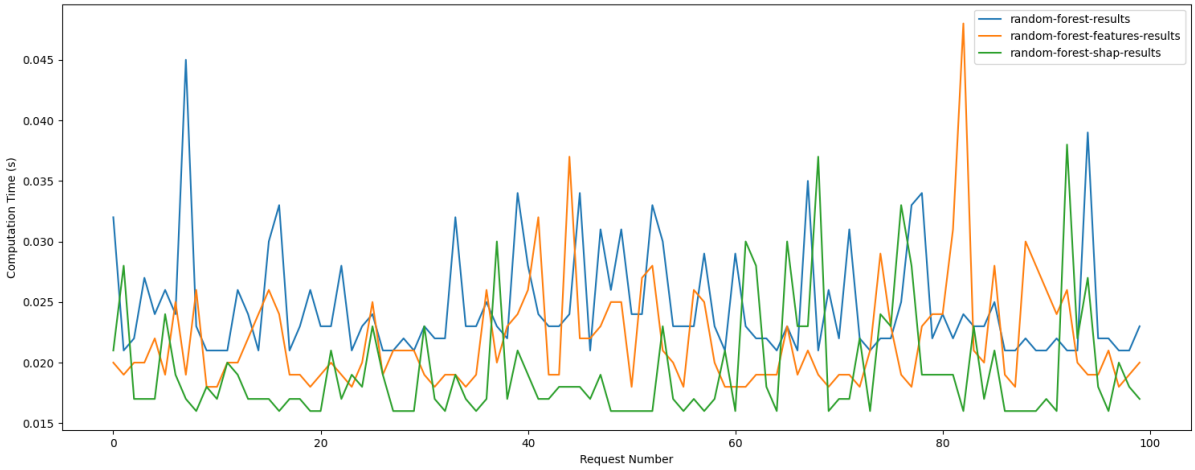

Figure 4.6: Random Forests computation times comparison.

Taking into consideration all of the factors, we now head to evaluate how this technique performs. Regarding the cost of data, it will be reduced as the processing of data and its ingestion will become faster, due to the dataset having a smaller number of features, its size will also reduce, hence C_{cost} will drop. In terms of the cost of the model itself, we can acknowledge an increase of its cost, as computing the Feature Selection implies that a maintenance ($C_{maintenance}$) was done to the model to make it more efficient. The issue here is that there are differences here, as the Random Feature Selector takes more time than the SHAP-based Feature Reduction, but ultimately both do not take a significant increase.

When it comes to the cost of inference, this is where it severely changes, as the Feature Selector can take up important features, and that will have an impact on the performance of the model, both in its ROC curve and score, as well as computation times will be longer, when compared to the other options. SHAP-based however, does not underperform drastically in regards of the default Random Forest, so we can safely assume that the computation cost stays roughly the same (the main factor from this comes from computation times).

So as a final judgement on the technique, we can safely say that while the Random Feature Selector is not beneficial to implement in production, SHAP-based Feature Reduction displays great metrics and cost efficiency to pose as an alternative for a classic Random Forest model.

Pros	**Cons**
Faster inference times	Small computation overhead needed, more noticeable in Random Feature Selector
Smaller domain of variables	Random Feature Selector can decrease accuracy

Table 4.3: Pros and Cons of Feature Reduction.

4.2.3 Parallel Computation

There are plenty of things to be done regarding the architecture to make it run code in parallel, but the aspect we are referring to in this section is centered on the ML models themselves. The goal for this was to study whether ML models would be faster if ran using Parallel Computation. The way it was implemented was by using Ray [1], a framework that allows programmers to use parallel computation in Python.

This is deployed and tested in a single machine, in which inside runs this code in parallel. By using the $@ray.remote$ keyword (see Listing A.10) we define a *task* for Ray to complete, as well as it allowing the library itself to automatically determine the number of workers and CPUs it will use of the machine to run the code in parallel. By default, each task will consume one core.

The central focus for this technique was for it to be able to improve the model serving of the framework, hence the results are evaluated mainly in the inference of the model. Figure 4.7 depicts a comparison in the computation times of using a regular Random Forest, over using a Random Forest by employing Ray. It is visible a considerate gain in terms of time, as it lowers on average 0.01784 seconds (around 88.24%).

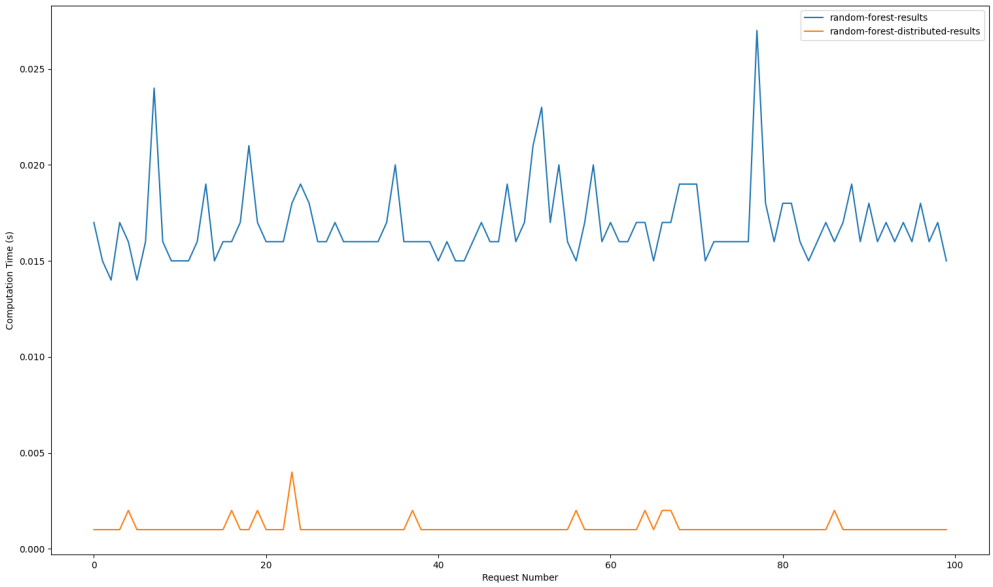

Figure 4.7: Default Random Forest versus Parallel computation times.

When looking at ML metrics (Figure 4.8), just by looking at the ROC curve that it produces, we can see that it does not have a significant drop of performance when compared to the sequential computation

[1] https://www.anyscale.com/ray-open-source, accessed on 20 Mar 2024

of a Random Forest (see Figure 4.2).

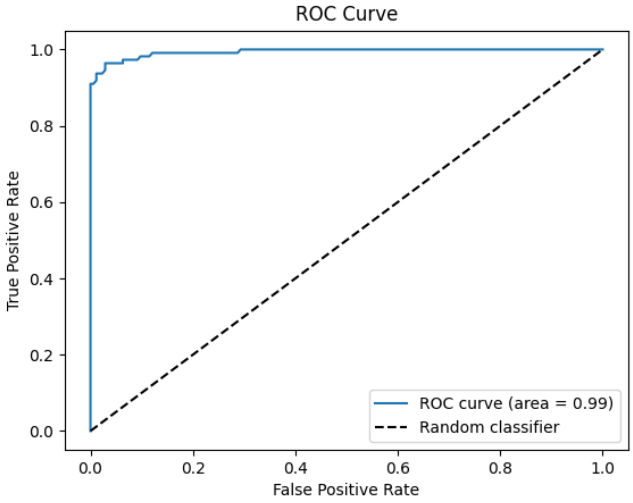

Figure 4.8: ROC curve of a Random Forest computed in parallel.

The same can be said for other models, to not only retain to Random Forests, the same time computation analysis was retained for Isolation Forests as well, as Figure 4.9 shows, the times also lower significantly, going down on average 0.01127 seconds (a decrease of 63.85%).

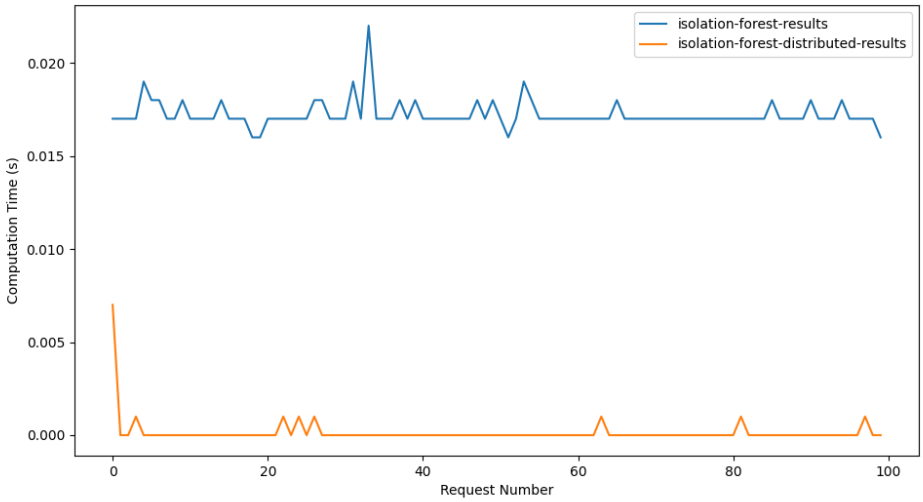

Figure 4.9: Default Isolation Forest versus Parallel computation times.

Besides this looking a pretty promising technique, there are a few aspects to take into consideration. Firstly, implementation-wise it is better to create an instance of a Random Forest per worker as Listing A.10 depicts. This was checked for prior to testing in production. Then, due to how the library is

incorporated, it needs a method called $ray.get$ (also shown in Listing A.10) which retrieves the results from the parallel environment. This adds a hefty overhead to the program if not using this on heavy models so that the gain in computation outweighs this.

With all of what was described and evaluated towards Parallel Computation, we can easily say that the only changes made to the Cost Model are on the $C_{inference}$ side, more specifically $C_{computation}$. This will mainly decrease those costs, if taken into account the overhead mentioned above. Therefore we can deem this technique as advantageous to apply to systems in production, where there are complex models and big data processing.

Pros	**Cons**
Faster inference times	Library specific overhead induced

Table 4.4: Pros and Cons of parallel processing.

4.2.4 Quantization - Deep NN

As mentioned in Chapter 3 for regular Deep NN we tested using static and dynamic quantization. Since for both static and dynamic quantization they are done without requiring any re-training of the model there wouldn't typically be any change made to the cost referring to models, but since quantizing still requires a small amount of time to prepare the weights, we can count as minimal increase in the cost of model changes. The model of choice to represent Deep NNs is a Multilayered-Perceptron (MLP) since it finds a great balance between model complexity and relevance in the industry.

Firstly in Figure 4.10 is displayed the confusion matrix of the default MLP model, for comparison purposes against the two forms of quantization of the MLP weights. Confusion matrices usually give insights on binary classifications, namely on whether the predictions are True Positives, True Negatives, False Negatives and False Positives, but since this model consists of 10 outputs the insights these provide are based on how well the model is able to predict correctly, and then what are the most usual mispredictions. As we can see in the confusion matrix the model performs well at labeling the inputs correctly, only having some wrong predictions, which do not induce abnormal behavior in the model's prediction capabilities.

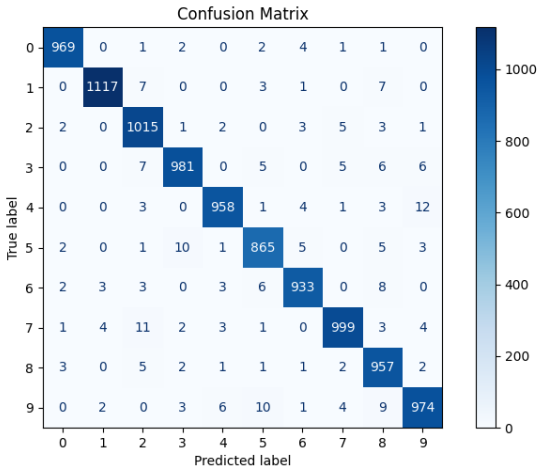

Figure 4.10: Default MLP confusion matrix.

Figure 4.11 shows us a plot of the computation times differences between a default MLP model, and then using static and dynamic quantization. It is visible that static quantization is faster than using a default model, with an average difference of 0.04823 seconds (a decrease of 35.54%), but dynamic quantization ends up being the fastest of the three with an average difference to the default MLP of 0.06035 seconds (44.48% less than the original model).

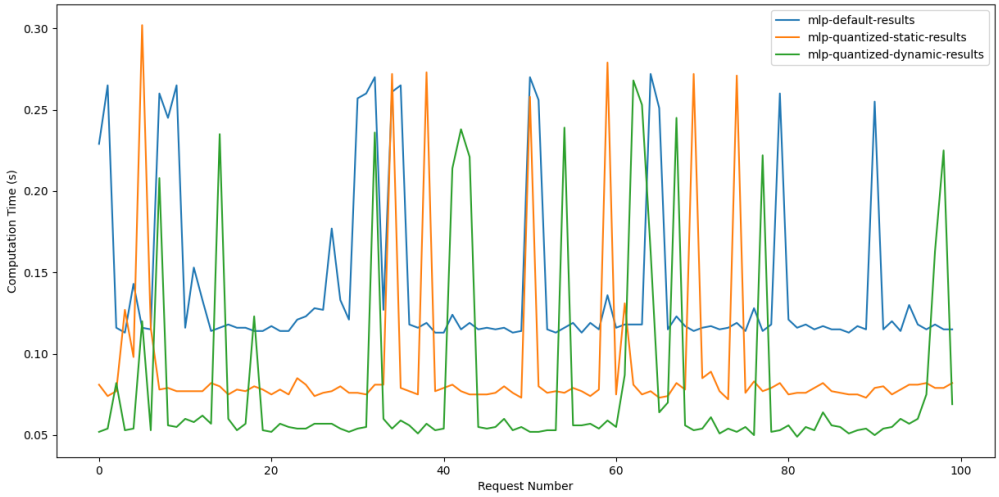

Figure 4.11: MLP computation times comparison.

Figure 4.12 and Figure 4.13 show the resulting confusion matrix of static quantization and dynamic quantization, respectively. We can see that both static and dynamic quantization do not differ much in their results when compared to the original model, but that dynamic quantization is very similar in the

results, while static quantization has more visible differences. To even add up on the results, the default MLP has an accuracy of 97.68%, which dynamic quantization also has, but static quantization drops to 95.24%.

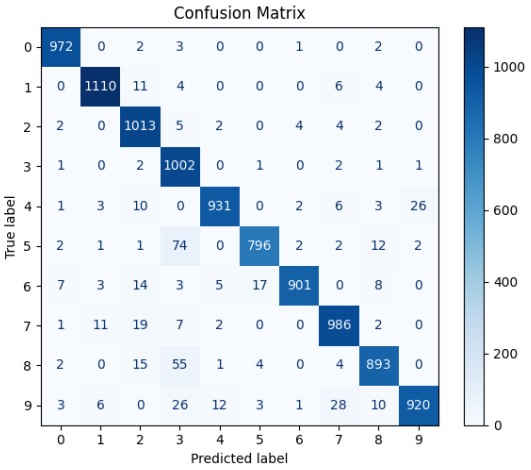

Figure 4.12: Confusion Matrix of MLP with static quantization.

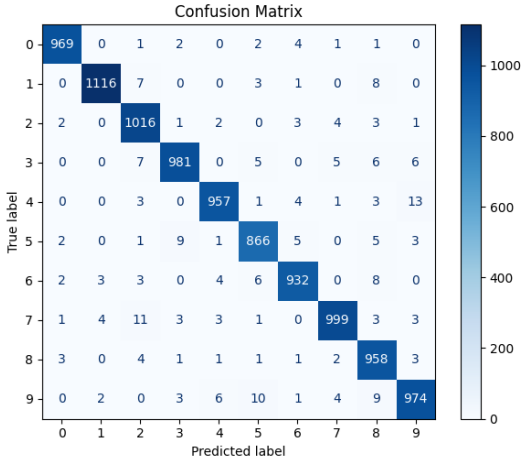

Figure 4.13: Confusion Matrix of MLP with dynamic quantization.

Needless to say, another aspect in which quantization is advantageous is space reduction. Since quantization converts the weights from one complex data type to a simpler one, it is logical that the size of the weight files also reduces. In the case in hands this is confirmed, as the static quantization weight file drops from 440.1kB to 116.6kB (a drop of 73.51% in size), and dynamic quantization goes to 115.0kB (a decrease of 73.87%).

Overall this technique looks very promising to use, as not only computation times reduce, the size

of the weights also go down significantly, all the while not having a relevant negative impact on the performance of the model, ML wise. In regards of cost, overall it will drop, as $C_{storage}$ drops, which leads to a decrease in C_{cost}. $C_{inference}$ will also be reduced as $C_{computation}$ lowers due to the quantized weights being faster in inference.

One could say that C_{model} would increase since to quantize its needs to run a separate script on the weights, but without requiring training, hence the cost will not increase drastically, and can even be overlooked.

As a final remark concerning this technique, it is encouraged to use it since it presents very good results. One could opt for either static or dynamic quantization, only choosing with a criteria of specific characteristic differences on the process of quantization, if the requirements for the model are not very strict. If this is not the case then dynamic quantization poses as the better option.

Pros	**Cons**
Faster inference times	Weights need to be quantized beforehand
Smaller weights size	Static Quantization has lower accuracy
Dynamic quantization has identical accuracy to the original model	

Table 4.5: Pros and Cons of Quantization of Deep NNs.

4.2.5 Quantization - YOLOv7

The quantization of Yolov7 follows the same principles of the quantization of Deep NNs, thus the changes on cost will theoretically be the same. The major change is that we will only perform dynamic quantization for this model. The reasoning comes from dynamic quantization for this model not needing to use training whilst static does. Since training would already give a great overhead of cost, and it from a hypothesis point of view, most cases it will not produce as good as a result as dynamic quantization if static quantization is not applied during training.

With this said, we now show some of the results of running YOLOv7 using the COCO[2] dataset. Figure 4.14 shows the confusion matrix resulting of running the model, and Figure 4.15 shows the F1 score of said run. It is important to mention that it had an average time of 100.7 milliseconds of inference time, and the size of the weights is of 75.6MB. Much of the analysis done on the confusion matrix of the Deep NN is identical to the one of YOLOv7. Since this is not a model for binary classification the explanation for the results of this confusion matrix is the same as before. The problem with only using a confusion matrix for YOLOv7 is that due to its complexity and number of labels it contains we can only see how true positives behave, hence it is also necessary to evalute its F1 curve, which evaluates the precision and recall of the model.

[2] https://cocodataset.org/, accessed on 17 Jul 2024

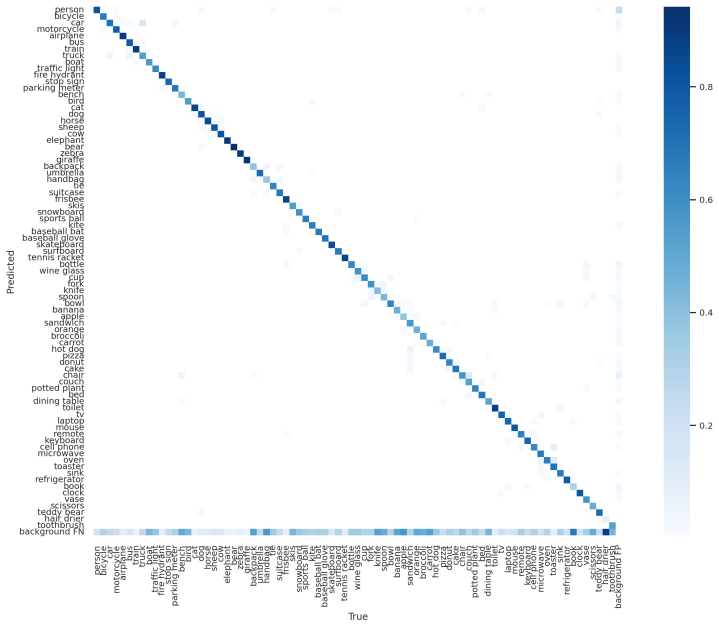

Figure 4.14: Confusion Matrix of YOLOv7.

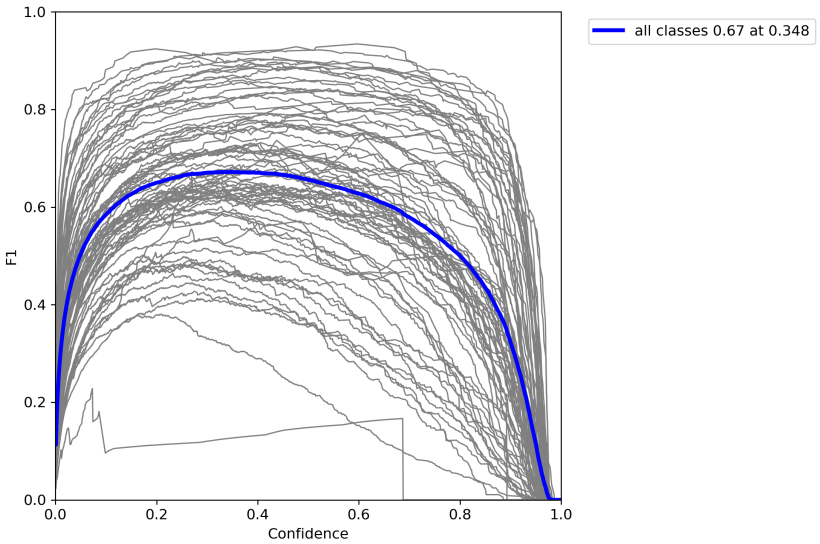

Figure 4.15: F1 curve of YOLOv7.

Now Figure 4.16 and Figure 4.17 show the confusion matrix and F1 score of the quantized YOLOv7, respectively. In terms of inference times it did an average of 21.0 milliseconds, so it has a difference of 79.7 milliseconds to the original model (almost a fifth of how long it takes for it to compute on average), and the quantized weights occupy 21.4 MB (decrease in size of 71.69%).

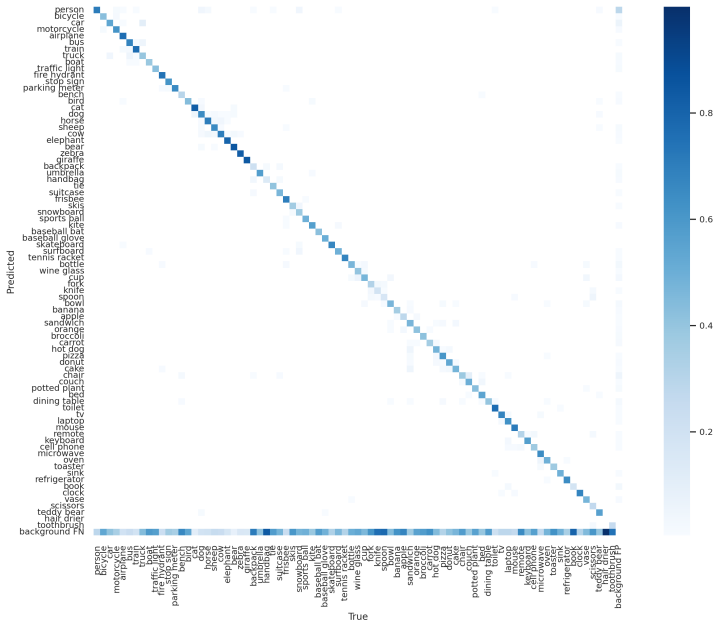

Figure 4.16: Confusion Matrix of YOLOv7 quantized.

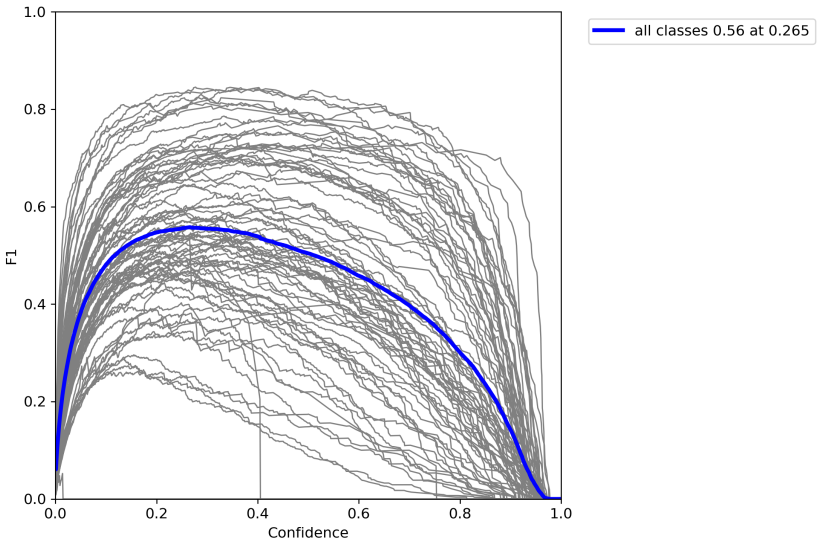

Figure 4.17: F1 curve of YOLOv7 quantized.

Overall we can see that this technique has its positives and negatives. The positives come of course from the decrease in cost that comes from this technique. Of course since computation times decrease $C_{computation}$, and, as a consequence, $C_{inference}$ also lowers. $C_{storage}$ also goes down since the file is smaller as well, hence C_{data} also drops. C_{model}, much like it was explained in the quantization of Deep NN, can have some minor increases, but not enough to be concerning.

In the end, what makes this technique not a must-have in ML-Ops systems is due to its performance

regarding ML metrics. As seen by the plots shown previously the quantized model does not perform well given a complex set of data, and many output classes (ten in this case).

Ultimately this technique should be employed, but carefully, as with complex datasets it will have its performance downgraded. One way to circumvent this could be by using a YOLOv7 with a low number of output classes, which makes the model more simple, so the drops in accuracy are not as accentuated as in a default YOLOv7 model.

Pros	Cons
Faster inference times	Weights need to be quantized beforehand
Smaller weights size	Very notable degradation in model performance

Table 4.6: Pros and Cons of the quantization of YOLOv7.

4.2.6 Communication Protocol - gRPC

gRPC comes as a technique that can be directly compared to HTTPS requests, so it strictly affects the inference, namely the time it takes to send a request and receive its results, thus affecting the costs of inference.

With this said, the simplest way to compare them is to gather data on the computation times of HTTPS and gRPC. Figure 4.18 depicts the results from those tests.

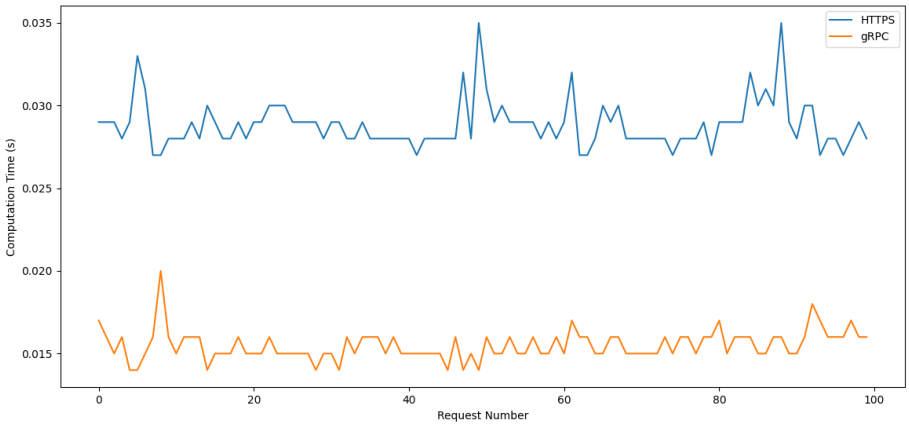

Figure 4.18: HTTPs versus gRPC protocols computation times.

As we can see by the results, gRPC is much faster than HTTPS, averaging a difference in computation of requests of 0.013 seconds (a drop of 41.93%). These results were obtained using the same model so that the only discriminating changes is the communication protocol.

When it comes to what this technique affects in the Cost Model, it will obviously affect $C_{inference}$, namely on $C_{computation}$. This does not affect the computation of the model directly, rather the computa-

tion of the request as a whole. One could also make a case that by employing gRPC the cost of Data will also increase, namely $C_{storage}$, since it now requires *protobuf* files and due to the implementation being in Python, it also needs *.py* files to create the handlers. But the truth is that those files occupy a mere total of 10kB, hence a negligible amount to be relevant to quantify.

In a final reflection, gRPC proves to be a viable method to reduce the computation time of requests, which also directly affects the costs of cloud services on the long-distance.

Pros	**Cons**
Faster request times	Small increase in file sizes, due to protobuf contracts

Table 4.7: Pros and Cons of using gRPC.

4.2.7 Batching Requests

Batching Requests in this work refers to the capability a framework can have by have multiple nodes serving the same request. Naturally, the prediction for this technique is to impact the inference in the Cost Model, more specifically the time it takes to fulfill a request, and the amount of requests done.

For the evaluation done, the batching is done by dividing the dataset into three equal parts, and then sending them to three different worker nodes. Figure 4.19 shows the computation times comparison of using one single worker to fulfill the request and using batching with three worker nodes.

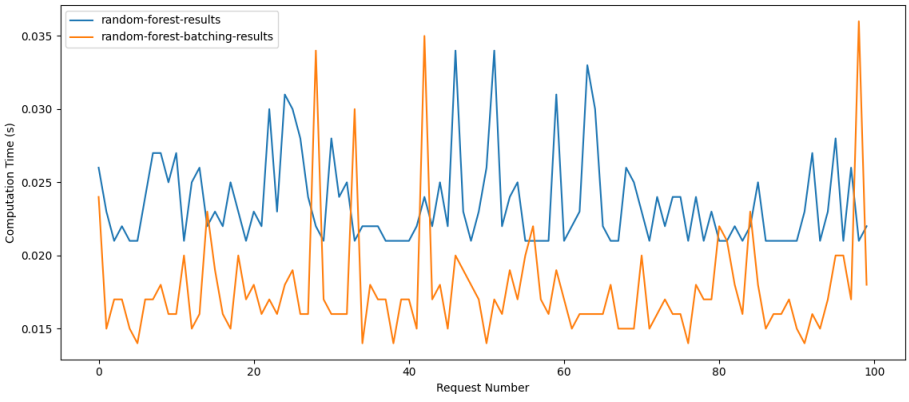

Figure 4.19: Default vs Batching computation times.

From analysing the figure we can see that there is a significant decrease in computation times, but since they are pretty unstable, the average difference in times comes to just about 0.0058 seconds (comes to a drop of about 23.2%). Not only this, but me must take into account that the amount of workers needed for the request to be fulfilled is tripled, hence the overhead for a single request increases.

By looking at this from a cost standpoint, we can clearly see that it will decrease $C_{computation}$. But the main factor which constrains this technique from only being beneficial is that the effective amount of requests triples, while the cost of computation does not reduce to one third of its original cost. Adding to the fact that having now using three worker nodes also increases the cost of scalability, $C_{inference}$ will ultimately increase in this case.

As a final note on the technique, it may seem that this technique is not worthy to implement in different ML-Ops systems, but thhe truth is that this technique takes advantage from hardware acceleration. In studies addressed during Chapter 2 (in the Related Systems section) we are presented which frameworks that perform since its goal is to implement in workers that are supplied with GPUs and deal with big loads of data. Therefore we do not discourage the usage of this technique, but would rather be cautious if the context is rich to implement it and observe gains.

Pros	**Cons**
Faster request times	Triple the overhead, due to multiple workers now fulfilling one request

Table 4.8: Pros and Cons of batching requests.

4.2.8 Surrogate Models

This technique specifically can be very broad in terms of what it impacts by applying it in production. To contextualize, in our Cost Model it can affect not only the cost of inference, but the cost of model as well. The cost of model is straightforward to understand, due to the time needed to train the surrogate model, as well as any maintenance or re-training needed to do if it starts to under perform.

The cost of inference on the other hand, comes using surrogate model in the way surrogate models were previously described in Chapter 2. For the purposes of this study, we focused on the surrogate model itself. Shadow models would add a very complex degree to this technique, which although valuable, would present fairly expectable results to apply the cost model formula.

With this said, like it was explained in Chapter 2, we used a KPLS as the surrogate model for an Isolation Forest. This was found the most suitable technique for Isolation Forests as they behave differently from Random Forests. From using it on the same dataset as the original model we can have a better understanding from the differences. Figure 4.20 depicts how the comparison of the original model with the final iteration of the surrogate model.

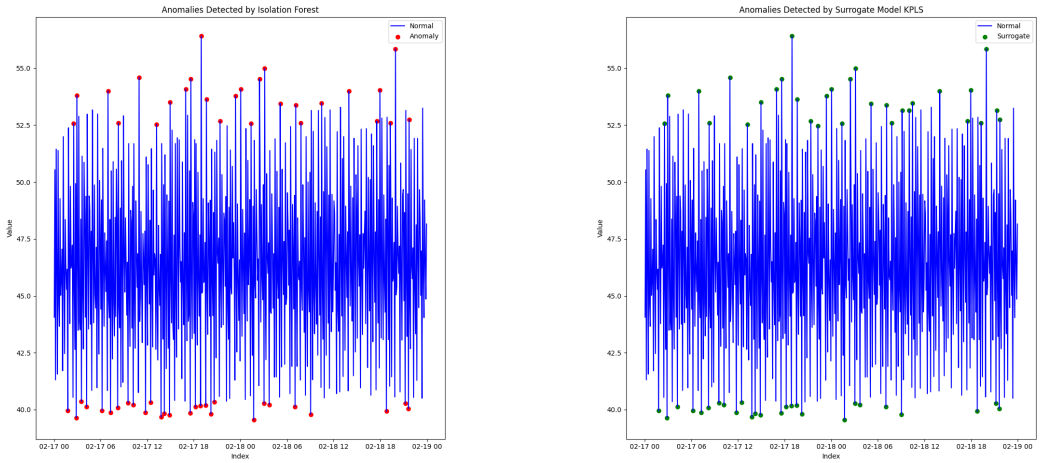

Figure 4.20: Original model vs surrogate anomaly detection comparison.

Figure 4.21 shows how the model Surrogate model performs, with a ROC curve based on the outputs of the Isolation Forest on the same test dataset. This is done this way since the *ground truth* values for the surrogate model come from the predictions made to the Isolation Forest, and as we are trying to approximate as much as possible the student model to its teacher this was the better way to analyse it. This means that the AUC should be as close to a squared shape as possible (since it should be approximate to a ROC value of 1.00).

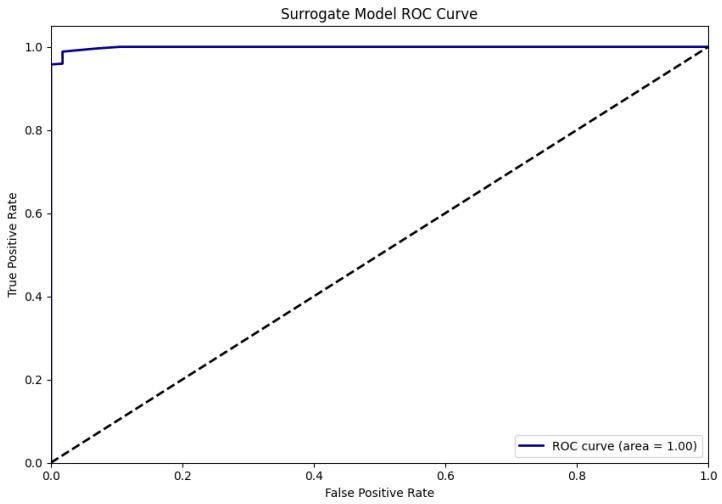

Figure 4.21: Original model vs surrogate model comparison.

How the surrogate model leverages against the original Isolation Forest in terms of computation times is described in Figure 4.22. These values indicate that although the surrogate model in theory

would be faster when compared to an Isolation Forest due to supposed lesser complexity, the original model still is able to compute faster at times, and with more stable computation times.

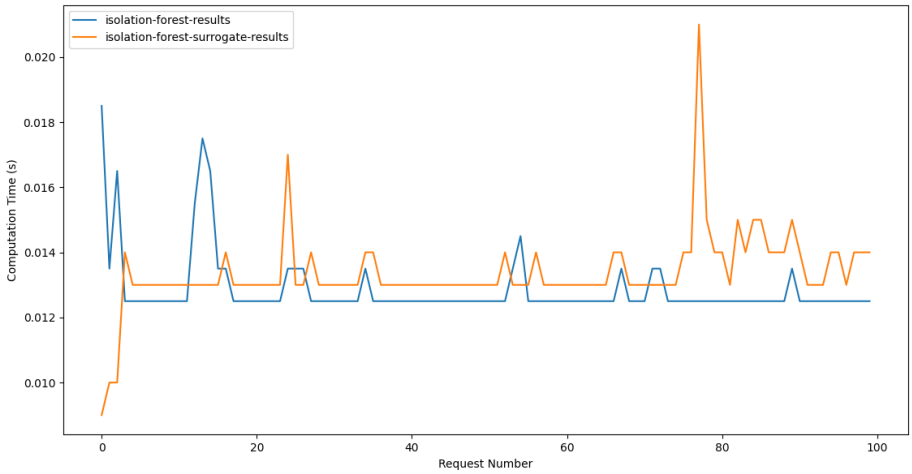

Figure 4.22: Original model vs surrogate model comparison.

Overall, this method in specific does not offer many strong points. The main strengths that come from using a Surrogate Model is that it offers a more comprehensible model (not necessarily less complex, as shown by the computation times) to have running in production without losing much performance ML wise, as well as occupying less space in memory, but only roughly about a 2MB (around 10% smaller in size) difference is detected. In terms of computation times, the surrogate model is on average 0.00049 seconds slower (an increase of 4.5%) that the default isolation forest model.

This being said, it does not offer much aside from this, as when it comes to computation times, where it is supposed to be its main strength, it does not out perform the original model. From a cost point of view, it will of course decrease C_{data} as the surrogate is about 10% lighter than the original model, affecting $C_{storage}$. When it comes to C_{model} it will of course raise, as the surrogate model requires training based on the outputs of the original model, so the increase comes from the direct raise in $C_{training}$. Not only this, but if the model is deployed to production, it will need maintenance (increasing $C_{maintenance}$ in that case) to understand if its performing to its standards, or it requires adjustments or even retraining altogether. Regarding $C_{inference}$ as shown by its performance metrics, it will have a subtle increase due to the increase in $C_{computation}$.

As a final judgement, this type of technique is better suited to more complex ML models. As it was already mentioned in Chapter 2, Isolation Forests have a near-linear complexity, so surrogate models would at the best case scenario only replicate the same costs the Isolation Forests have in production, barring training costs. The better course of action for a model that has this level of simplicity for its ef-

fectiveness to reach its goal, techniques already mentioned such as Data Transformation and Ingestion, as well as Parallel Computation are better techniques to apply than Surrogate Models.

Pros	**Cons**
Simplified model	Requires training
Occupies less space in memory	Similar, or slower times than the original model
Very similar results in terms of accuracy	

Table 4.9: Pros and Cons of surrogate models.

4.3 Summary

In this Chapter the solution that was proposed earlier was evaluated. First it was given a contextualized example of real production expenses. This is important to understand how the techniques studied can make an impact in the costs of running cloud ML services.

Then came the task to test and evaluate every technique that was rendered fit to undergo this task. In every task we reiterate what it consists of, then give a simple baseline of the original state of that specific aspect of the framework, whether it is a ML model or ML-Ops part.

Next we evaluate the results of the technique applied to the framework. We compare side by side the original status and the changes made to it, for it to become a simpler and more direct comparison, with no adding layers different from one another. This comparison also implies calculating differences in variables that directly affect cost changes, more notably times, and space/data size.

Finally, we undergo the task to understand how these changes reflect on cost themselves, and if the impact on the total cost of just applying that single technique is positive or negative, and with that in mind conclude if the technique is useful or not. In the final description of each technique evaluation a small Pros and Cons table is shown to summarize the results obtained.

5

Conclusion

Contents

In this work we presented Smart-ML, a framework which is capable of monitoring ML models in the cloud with the intuition to implement and monitor cost-effective models. We presented the relevancy of the problem, and how it covers so many important domains, namely Machine Learning and Cloud Computation in technology and society nowadays. The importance derives from the broad spectrum of appliances that ML can have and how can companies maintain their ML services available to the ever growing demand while keeping those services sustainable. Adding to this, this study pretends to collect many different approaches to cost-reduction in Machine Learning Operations, into one single framework much like a review of several techniques gathered into a centralized piece of code.

The work featured a Chapter exclusively for Background due to the diverse fields of knowledge featured in this work which calls for a brief explanation of the many topics addressed. In its length we go over concepts which incur both in Cloud Computation and Machine Learning. Subjects which are associated with cost-reduction like scaling cloud environment and different types of cloud services are explained, much like several ML models that are relevant to be presented and explained, which see plenty of usage in important industry sectors, for instance Healthcare, Agriculture and Manufacturing. Besides, it also highlights several Related Systems, which not only serve as inspiration for the work, also display the many forms ML-Ops systems, or even ML models can suffer alterations to achieve a specified goal towards better efficiency.

In the Proposed Solution, we present the architecture and studies done for the work developed. Firstly we presented how the first studies to filter the ML model cost reduction techniques were systematized, in an effort to make this process as swift as possible. This made it possible to filter out models and techniques to be studied, so that we end upon the techniques to evaluate. Then we explained the architecture of the framework to be used where we will conduct the testing of all of the techniques, to understand how each of the techniques work, and its purpose in the framework. Finally, it was presented a Cost Model, which contains cost formulas to take into account during the evaluation of each technique, since a single technique can affect the three main areas of impact: Data, Model, and Inference. Then, it was explained how the decision criteria to evaluate whether the impact of a technique is positive or negative, by using some simple to understand and use decision algorithms.

Lastly the work contains the Evaluation Chapter where we take on the test & evaluation task of every technique described in the previous Chapter. In this section we go over all the results obtained from each technique enunciated which are found relevant to make a decision on it being beneficial or not cost-wise. This means we study changes and comparisons between computation times, files sizes, as well as ML oriented techniques such as ROC curve, F1 score and model complexity. With all of this data gathered, the information retained becomes rich enough to apply the cost formulas created earlier, and then the algorithms of decision to make the final assessment on the technique. Lastly on the end of each technique evaluation it is also featured a pros and cons table to summarize the experiments and

subsequent takeaways.

5.1 Future Work

With this work done, several options are opened to study. One of the more direct ones is to layer techniques, or combining them, to assess how could this added complexity behave in terms of cost. The other studies come from the fact that for this work we were limited to a local machine to test the results, so an extension of this could came by testing this in a bigger, and more powerful environment with multiple machines.

This opens up a lot of possibilities, ranging from becoming possible to test hardware acceleration techniques (exploiting GPUs more specifically), to exploring more how cloud could also suffer changes to make the system more cost-efficient. This implies using concepts like load balancing, auto-scaling, or even try ML oriented techniques like surrogate models, but with the original model as backup. Taking a step further, besides having models for predictive maintenance as agents to swap a underperforming cost-efficient model with its original counterpart, we could also have Artificial Intelligent agents running in algorithms based on this work to gather and apply techniques in real-time into a framework. By studying the impact of Cloud Computation further works would have more complexity, but if positive results emerge from them, then it is sure that cost would be further decrease for these types of systems.

Bibliography

[1] W.-T. Tsai, X. Sun, and J. Balasooriya, "Service-oriented cloud computing architecture," in *2010 Seventh International Conference on Information Technology: New Generations*, 2010, pp. 684–689.

[2] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015, accessed on 20 Dec 2023. [Online]. Available: http://arxiv.org/abs/1511.08458

[3] Amazon, "Debiasing ai using amazon sagemaker," 2019, acessed on 30 Nov 2023. [Online]. Available: https://aws.amazon.com/sagemaker/

[4] C. Zhang, M. Yu, W. Wang, and F. Yan, "Enabling cost-effective, slo-aware machine learning inference serving on public cloud," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1765–1779, 2022.

[5] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[6] Y. Hu, R. Ghosh, and R. Govindan, "Scrooge," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021.

[7] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27–33.

[8] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 162–169.

[9] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ML serving," *CoRR*, vol. abs/1712.06139, 2017. [Online]. Available: http://arxiv.org/abs/1712.06139

[10] N. R. Herbst, S. Kounev, and R. Reussner, “Elasticity in cloud computing: What it is, and what it is not,” in *10th International Conference on Autonomic Computing (ICAC 13)*, Jun. 2013, pp. 23–27.

[11] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[12] J. Simão and L. Veiga, “Qoe-jvm: An adaptive and resource-aware java runtime for cloud computing,” in *On the Move to Meaningful Internet Systems: OTM 2012*, R. Meersman, H. Panetto, T. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I. F. Cruz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 566–583.

[13] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, “Auto-scaling vnfs using machine learning to improve qos and reduce cost,” in *2018 IEEE International Conference on Communications (ICC)*, 2018.

[14] Kubernetes, “Kubernetes pod autoscale,” 2023, acessed on 10 Dec 2023. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[15] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, “A survey of machine learning for big data processing,” *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, 2016.

[16] I. Muhammad and Z. Yan, “Supervised machine learning approaches: A survey,” *ICTACT Journal on Soft Computing*, vol. 05, no. 03, p. 946–952, 2015.

[17] H. Bhavsar and M. H. Panchal, “A review on support vector machine for data classification,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 1, no. 10, pp. 185–189, 2012.

[18] W. S. Noble, “What is a support vector machine?” *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[19] R. Xu and D. Wunsch, “Survey of clustering algorithms,” in *IEEE Transactions on Neural Networks*, vol. 16, no. 3, 2005, pp. 645–678.

[20] S. Tsang, B. Kao, K. Y. Yip, W.-S. Ho, and S. D. Lee, “Decision trees for uncertain data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 64–78, 2011.

[21] M. Schonlau and R. Y. Zou, “The random forest algorithm for statistical learning,” *The Stata Journal*, vol. 20, no. 1, pp. 3–29, 2020.

[22] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, “An ensemble random forest algorithm for insurance big data analysis,” *IEEE Access*, vol. 5, pp. 16 568–16 575, 2017.

[23] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[24] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, “Multilayer perceptron and neural networks,” *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009.

[25] Y. Yu, X. Si, C. Hu, and J. Zhang, “A review of recurrent neural networks: Lstm cells and network architectures,” *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[26] A. Rajkomar, J. Dean, and I. Kohane, “Machine learning in medicine,” *New England Journal of Medicine*, vol. 380, no. 14, p. 1347–1358, 2019.

[27] G. Chugh, S. Kumar, and N. Singh, “Survey on machine learning and deep learning applications in breast cancer diagnosis,” *Cognitive Computation*, vol. 13, no. 6, p. 1451–1470, 2021.

[28] A.-A. Nahid, M. A. Mehrabi, and Y. Kong, “Histopathological breast cancer image classification by deep neural network techniques guided by local clustering,” *BioMed Research International*, vol. 2018, p. 1–20, 2018.

[29] V. H. S. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. C. Dias, and M. P. Guimarães, “Machine learning applied to software testing: A systematic mapping study,” *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–1212, 2019.

[30] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[31] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, “A review of machine learning and iot in smart transportation,” *Future Internet*, vol. 11, no. 4, p. 94, 2019.

[32] Y. Tian and L. Pan, “Predicting short-term traffic flow by long short-term memory recurrent neural network,” in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, 2015, pp. 153–158.

[33] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, “Machine learning in manufacturing: advantages, challenges, and applications,” *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.

[34] U. M. Paturi and S. Cheruku, “Application and performance of machine learning techniques in manufacturing sector from the past two decades: A review,” *Materials Today: Proceedings*, vol. 38, p. 2392–2401, 2021.

[35] K. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, “Machine learning in agriculture: A review,” *Sensors*, vol. 18, no. 8, p. 2674, 2018.

[36] J. Diaz-De-Arcaya, A. I. Torre-Bastida, G. Zárate, R. Miñón, and A. Almeida, "A joint study of the challenges, opportunities, and roadmap of mlops and aiops: A systematic survey," *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–30, 2023.

[37] D. A. Tamburri, "Sustainable mlops: Trends and challenges," in *2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC)*. IEEE, 2020, pp. 17–23.

[38] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[39] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," in *Proceedings of the VLDB Endowment*, vol. 4, no. 11, 2011, p. 1111–1122.

[40] S. Esteves, H. Galhardas, and L. Veiga, "Adaptive execution of continuous and data-intensive workflows with machine learning," in *Proceedings of the 19th International Middleware Conference, Middleware 2018, Rennes, France, December 10-14, 2018*, P. Ferreira and L. Shrira, Eds. ACM, 2018, pp. 239–252. [Online]. Available: https://doi.org/10.1145/3274808.3274827

[41] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[42] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 2016.

[43] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[44] M. E. Coimbra, S. Esteves, A. P. Francisco, and L. Veiga, "Veilgraph: incremental graph stream processing," *J. Big Data*, vol. 9, no. 1, p. 23, 2022. [Online]. Available: https://doi.org/10.1186/s40537-022-00565-8

[45] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[46] D. Plase, L. Niedrite, and R. Taranovs, "A comparison of hdfs compact data formats: Avro versus parquet," *Mokslas–Lietuvos ateitis/Science–Future of Lithuania*, vol. 9, no. 3, pp. 267–276, 2017.

[47] B. Morais, M. E. Coimbra, and L. Veiga, "Pk-graph: Partitioned k^2-trees to enable compact and dynamic graphs in sparkgraphx," in *Cooperative Information Systems - 28th International*

Conference, CoopIS 2022, Bozen-Bolzano, Italy, October 4-7, 2022, Proceedings, ser. Lecture Notes in Computer Science, M. Sellami, P. Ceravolo, H. A. Reijers, W. Gaaloul, and H. Panetto, Eds., vol. 13591. Springer, 2022, pp. 149–167. [Online]. Available: https://doi.org/10.1007/978-3-031-17834-4_9

[48] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, no. 2. PMLR, 17–19 Jun 2013, pp. 199–207.

[49] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[50] M. Phuong and C. Lampert, "Towards understanding knowledge distillation," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 5142–5151.

[51] G. K. Nayak, K. R. Mopuri, V. Shaj, V. B. Radhakrishnan, and A. Chakraborty, "Zero-shot knowledge distillation in deep networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 4743–4751.

[52] R. Alizadeh, J. K. Allen, and F. Mistree, "Managing computational complexity using surrogate models: a critical review," *Research in Engineering Design*, vol. 31, no. 3, pp. 275–298, 2020.

[53] A. Iranmehr, H. Masnadi-Shirazi, and N. Vasconcelos, "Cost-sensitive support vector machines," *Neurocomputing*, vol. 343, pp. 50–64, 2019.

[54] K.-M. Lin and C.-J. Lin, "A study on reduced support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1449–1459, 2003.

[55] B. Li, S. Samsi, V. Gadepally, and D. Tiwari, "Clover: Toward sustainable ai with carbon-aware machine learning inference service," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–15.

[56] E. Florian, F. Sgarbossa, and I. Zennaro, "Machine learning-based predictive maintenance: A cost-oriented model for implementation," *International Journal of Production Economics*, vol. 236, p. 108114, 2021.

[57] C. N. Prometheus, "Prometheus - monitoring system & time series database," 2014, acessed on 5 Jan 2024. [Online]. Available: https://prometheus.io/

[58] G. Labs, "Grafana," 2014, acessed on 5 Jan 2024. [Online]. Available: https://grafana.com/docs/

[59] S. T. Limited, “Seldon,” Nov 2023, acessed on 5 Jan 2024. [Online]. Available: https://www.seldon.io/

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol–http/1.1,” W3C/MIT, Tech. Rep., 1999.

[61] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, “Measuring {HTTPS} adoption on the web,” in *26th USENIX security symposium (USENIX security 17)*, 2017, pp. 1323–1338.

[62] Google, “grpc,” 2024, accessed on 30 Aug 2024. [Online]. Available: https://grpc.io/

[63] C. Z. Shichao Zhang and Q. Yang, “Data preparation for data mining,” *Applied Artificial Intelligence*, vol. 17, no. 5-6, pp. 375–381, 2003.

[64] M. Frye and R. H. Schmitt, “Structured data preparation pipeline for machine learning-applications in production,” *17th IMEKO TC*, vol. 10, pp. 241–246, 2020.

[65] Z. Zhou and G. Hooker, “Unbiased measurement of feature importance in tree-based methods,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 2, pp. 1–21, 2021.

[66] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” *Advances in neural information processing systems*, vol. 30, 2017.

[67] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, “A review of yolo algorithm developments,” *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022, the 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922001363

[68] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 7464–7475.

[69] Microsoft, “Ncas t4 v3-series - azure virtual machines,” 2023, acessed on 5 Jan 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/nct4-v3-series

A

Code of Project

In this chapter is where all pieces of code relevant to be mentioned, or simply displayed for clarification purposes are contained. This contains mostly code that is oriented for the implementation of the work, as ML specific model code are not be deeply detailed, this chapter focus on how it was implemented to be used on the framework.

Listing A.1: Protobuf contracts

```
syntax = "proto3";
package worker;

service WorkerService {
  rpc RandomForestRequest (RfRequest) returns (RandomForestResponse);
  rpc IsolationForestRequest (IsoRequest) returns (IsolationForestResponse);
  rpc MLPRequest (MlpRequest) returns (MLPResponse);
}
message RfRequest {
  string model = 1;
  repeated RfRequestDict dataset = 2;
}

message RandomForestResponse {
  string string = 1;
  repeated RFResults results = 2;
}

message IsoRequest {
  string model = 1;
  repeated IsoRequestDict dataset = 2;
}

message IsolationForestResponse {
  string string = 1;
  repeated IsoResponseDict results = 2;
}

message MlpRequest {
  string model = 1;
  repeated MlpImg records = 2;
}

message MLPResponse {
  string string = 1;
  repeated int32 results = 2;
}
```

We didn't define the message types `RfRequestDict`, `RFResults`, `IsoRequestDict`, `IsoResponseDict`, `IsoResponseDict`, `MlpImg` since these are implementation and dataset specific data structures, which can change, with the change in data used for the requests, hence deemed not necessary to represent in the code.

Here is the Dockerfile that creates the Docker image which the containers use, as well as the Kubernetes Pods.

Listing A.2: Dockerfile

```
FROM python:3.8.10

WORKDIR /app

COPY worker-node.py /app
COPY isolationforests.py /app
COPY randomforests.py /app
COPY mlpdnn.py /app
COPY requirements.txt /app
COPY dataset/wisconsin.csv /app
COPY dataset/ec2_cpu_utilization_5f5533.csv /app
COPY dataset/mnist.npz /app
COPY mlp_weights.pth /app
COPY mlp_dynamic_quantized_weights.pth /app
COPY mlp_static_quantization_weights.pth /app

RUN pip install --no-cache-dir -r requirements.txt

ENV PORT=3000

CMD ["sh", "-c", "python worker-node.py -p $PORT"]
EXPOSE $PORT
```

Following are the files that help setup the Kubernetes Pods environment, as well as the service to make the Pods communicate using their ports.

Listing A.3: Pods launcher YAML file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-testing-deployment-1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: model-testing-1
  template:
    metadata:
      labels:
        app: model-testing-1
    spec:
      containers:
      - name: model-testing
        image: whoisginja/model_testing:v1
        ports:
        - containerPort: 3000
        env:
        - name: PORT
          value: "3000"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-testing-deployment-2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: model-testing-2
  template:
```

```
    metadata:
      labels:
        app: model-testing-2
    spec:
      containers:
      - name: model-testing
        image: whoisginja/model_testing:v1
        ports:
        - containerPort: 3001
        env:
        - name: PORT
          value: "3001"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-testing-deployment-3
spec:
  replicas: 1
  selector:
    matchLabels:
      app: model-testing-3
  template:
    metadata:
      labels:
        app: model-testing-3
    spec:
      containers:
      - name: model-testing
        image: whoisginja/model_testing:v1
        ports:
        - containerPort: 3002
        env:
        - name: PORT
          value: "3002"
```

Listing A.4: Service launcher YAML file

```yaml
apiVersion: v1
kind: Service
metadata:
  name: model-testing-service-1
spec:
  selector:
    app: model-testing-1
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: model-testing-service-2
spec:
  selector:
    app: model-testing-2
  ports:
    - protocol: TCP
      port: 3001
      targetPort: 3001
---
apiVersion: v1
kind: Service
metadata:
  name: model-testing-service-3
spec:
  selector:
    app: model-testing-3
  ports:
    - protocol: TCP
      port: 3002
      targetPort: 3002
```

Besides this YAML files, there is also a YAML file to be able to launch the Prometheus and Grafana services using the Docker Compose tool.

Listing A.5: Docker Compose YAML file

```
version: '3'
services:
  prometheus:
    image: prom/prometheus
    volumes:
      - "./prometheus.yml:/etc/prometheus/prometheus.yml"
    ports:
      - 9090:9090


  grafana:
    image: grafana/grafana
    ports:
      - 3004:3000
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-storage:/var/lib/grafana


  model_testing_1:
    image: whoisginja/model_testing:v1
    ports:
      - 3000:3000
      - 9100:9100

  model_testing_2:
    image: whoisginja/model_testing:v1
    ports:
      - 3001:3000
      - 9101:9100

  model_testing_3:
    image: whoisginja/model_testing:v1
    ports:
      - 3002:3000
      - 9102:9100
```

```
38
39 volumes:
40   grafana-storage:
```

Following the Prometheus YAML file that ensures that Prometheus is able to scrape metrics from each Pod that is launched.

Listing A.6: prometheus.yml YAML file

```
1  global:
2    scrape_interval: 10s
3  scrape_configs:
4   - job_name: prometheus
5     static_configs:
6      - targets:
7          - prometheus:9090
8   - job_name: model_testing_1
9     static_configs:
10     - targets:
11         - model_testing_1:9100
12  - job_name: model_testing_2
13    static_configs:
14     - targets:
15         - model_testing_2:9100
16  - job_name: model_testing_3
17    static_configs:
18     - targets:
19         - model_testing_3:9100
```

Now we will describe in pseudo-code how the framework is implemented to receive and send requests, as well as how the models are setup to process the requests.

Listing A.7: Python client code

```
def sendRequest(payload):
    response = send(payload)
    print(response.results)

def modelRequest(data, model_measure):
    data_processed = prepareData(data)

    #format to desirable data format for communication protocol
    payload = [model_meaure, data_processed]
    sendRequest(payload)
```

Each model then has its specific `modelRequest` function, with data processing and payloads according to each of them.

Listing A.8: Python worker code

```
class ModelRequestsHandler():
    def __init__():
        self.models = loadModels()

    def receiveModelRequest(payload):
        model_measure = payload.model_measure
        data = payload.data

        model = getModel(model_measure)

        data_processed = prepare_data(data) #convert the data to same
            structure used in ML model
        results = model.process(data_processed)

        payload = [model_measure, results]
        return payload

```

```
17 if __name__ = "__main__":
18     serve()
```

Similar to the client code, the worker node also has this general code structure for every model, each having its specific way to prepare the data, and the response payload.

Analogously to the previous code shown, all the models, as well as the ones with measures applied, follow the same code structure, presented below in Listing. This is done so all the code is uniform and easier to deploy and test.

Listing A.9: Python ML model code

```
1 class MLModel():
2     def __init__():
3         self.model = prepareModel()
4 
5     def process(data):
6         results = self.model.predict(data)
7 
8         #for metrics purposes
9         results_probability = self.model.predictProba(data)
```

This is the base code from implementing Parallel Computation of Random Forests using Ray. Inside the function is where the new instance of a Random Forest, and to obtain the results of the prediction from every worker into a single structure it uses the $ray.get$ method.

Listing A.10: Ray Python implementation example

```
1 import ray
2 
3 @ray.remote
4 def train_random_forest(X_train, y_train, X_test):
5     rf = RandomForestClassifier(n_estimators=100, max_depth=42)
6     rf.fit(X_train, y_train)
7     y_pred = rf.predict(X_test)
8     y_pred_prob = rf.predict_proba(X_test)[:, 1]
9     return rf, y_pred, y_pred_prob
10 
11 
```

```
ray.init(ignore_reinit_error=True)
rf_ref = train_random_forest.remote(X_train, y_train, X_test)

# Get the results
rf, y_pred, y_pred_prob = ray.get(rf_ref)
```