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

JOÃO RAMALHO, Instituto Superior Técnico, Portugal

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, having a deeper focus on the latter. 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 each case requires changes to the way Smart-ML applies, according to its necessities.

1 Introduction

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] [Amazon 2019; Zhang et al. 2022], the main challenge that arises is how can developers reduce costs by applying direct changes to ML models.

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.

In this work we present Smart-ML, a framework that contains ML-models and cost reduction measures applied to the framework, which is presented in Section 3. These measures 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 measures and to facilitate in the evaluation of each measure.

2 Related Work

Machine Learning embodies a predominant portion of this work, and many fields have utilised its capabilities to great extent, therefore we state 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. [Desislavov et al. 2023] 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.

[1] https://pjreddie.com/darknet/yolo/, accessed on 10 Set 2024

Author's Contact Information: João Ramalho, Instituto Superior Técnico, Lisboa, Portugal, joao.ginja.ramalho@tecnico.ulisboa.pt.

[2] https://azure.microsoft.com/en-us/products/machine-learning, accessed on 8 Nov 2023.

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.

Hyperparameter training. This type of ML models training is one of the latter techniques and Bardenet et al. [Bardenet et al. 2013] 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. [Zhu and Gupta 2017] 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. [Nayak et al. 2019; Phuong and Lampert 2019], 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.

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 [Alizadeh et al. 2020].

Cost-Sensitive SVM. Work from Iranmehr et al. [Iranmehr et al. 2019] 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. [Lin and Lin 2003] also presented a different version of a cost-oriented SVM, named Reduced Support Vector Machine (RSVM) 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.

Model Ark (MArk). Zhang et al. [Zhang et al. 2022] 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. [Ali et al. 2020] 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.) [Hu et al. 2021] 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. [Li et al. 2023] 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. [Florian et al. 2021] 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.

Finally it is necessary to mention that recently frameworks were created to facilitate the deployment of ML models such as Tensorflow [3], scikit-learn [4] and Pytorch [5]. 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.

3 Solution

In this Section 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 models were chosen taking into account industry sectors that use ML models in production, or even have the potential to incorporate ML to great extent. The sectors taken into account are Healthcare, Transportation, Manufacturing, Agriculture and Software Development.

[3] https://www.tensorflow.org/, accessed on 20 Mar 2024
[4] https://scikit-learn.org/, accessed on 20 Mar 2024
[5] https://pytorch.org/, accessed on 20 Mar 2024

The reason for choosing these five sectors were essentially due to relevancy, and abundance of usage of ML in each sector. In essence, 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.

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.

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.

Table 1 sums up the results from applying the research and analysis tasks. 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 1. Table of decision based on the systematic tasks.

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 [Prometheus 2014] offers great monitoring services, which are also capable of storing these metrics, and then be queried when it is needed. Then Grafana [Labs 2014] can be used to create dashboards, as it is a great data visualization tool.

3.2.1 Distributed Architecture. 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.

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, as well as launching multiple workers in a single machine 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.

3.2.2 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 [Fielding et al. 1999] 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 [Felt et al. 2017]. 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 [Google 2024] 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 [6]. 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.

3.3 Techniques Overview

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 1, 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.

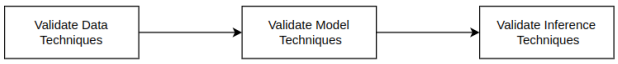

Fig. 1. General flow of technique validation.

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 2.

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

Table 2. Cost Reduction Techniques Studied.

All of these techniques derive from the study which used the tasks made earlier, as well as gathering other techniques which are not ML centered, such as Data related techniques, or parallel computation.

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

3.4 Cost Modelling

Now 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.

Keep in mind that although there are many domains that can define cost in a sense, like space, time, ML metrics and hardware, these do not necessiraly reflect on cost with the end-goals that we have settled for this system, but nonetheless very important to understand as they can be evaluated, or even be part of a larger composition that will define cost.

3.4.1 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**.

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.

3.5 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})

$$C_{data} = C_{storage} + C_{processing} + C_{ingestion} \tag{1}$$

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

Cost Model. (C_{model})

$$C_{model} = C_{training} + C_{retraining} + C_{maintenance} \tag{2}$$

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

Inference Cost. ($C_{inference}$)

$$C_{inference} = C_{compute} \times N_{requests} + C_{scaling} \tag{3}$$

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:

$$T_{cost} = C_{data} + C_{model} + C_{inference} \tag{4}$$

3.6 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.

3.6.1 Decision Algorithms. All of the decisions made will be based on the three main domains that described Cost in the previous Section. 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. We can assume that all of the changes and evaluation is done based on what Algorithm 1 shows, and that *costChanges* will be normally be a negative value. 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. 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 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 2.

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.

Algorithm 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()
```

Cost of Inference. The processing of the computations of the cost of inference can be seen in Algorithm 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.

Finally, the new cost of inference is computed by adding the product and the cost of scaling and then checked if its lower than the previous value of cost.

Algorithm 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
        if C_cost < oldC_cost then
            acceptchange
        else
            discardchange
```

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

Algorithm 4: Cost of Inference Decision Process

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

4 Evaluation

In this Section we go over some of the techniques applied to the framework. For the results shown we try to have at least one that covers each module: Data Techniques, Model Techniques and Inference Techniques.

4.1 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 measures, we have two similar in theory hypothesis: a Random Feature Selector and a SHAP-based. 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.

The principal actions done in the framework 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 subsequently transformed to a data loader for better processing.

These transformations overall allow for an increase to the cost of processing of data ($C_{processing}$), but the purpose of this transformations is to facilitate the computations done by the models.

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 2 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.

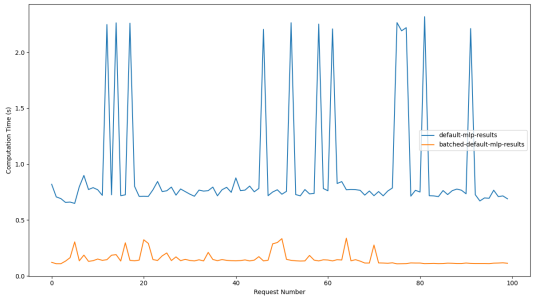

Fig. 2. Comparison of computation times of MLP model with batching.

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.

4.2 Feature Importance/Selection - Clustering

As aforementioned in this Section, 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 Random Feature Selector, which randomizes the labels to be used across multiple iterations to compute the most important features. 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.

Figure 3 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.

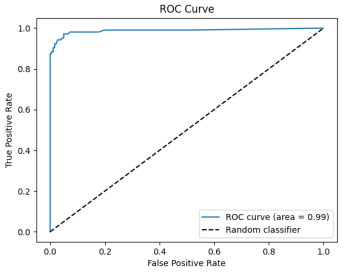

Fig. 3. ROC curve of a classic Random Forest model.

With this said, the ROC curve it produces is the one described by the left plot of Figure 5. 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.

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.

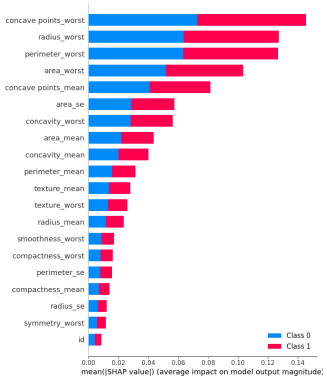

Fig. 4. Results of SHAP-based Feature Importance.

Then, the next step was to filter out the less relevant features by order out of the dataset. As mentioned 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 6. 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.

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 8 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.

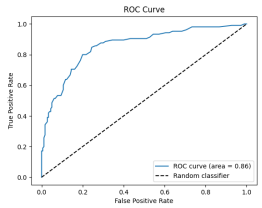

Fig. 5. ROC-Curve of Random Feature Selector.

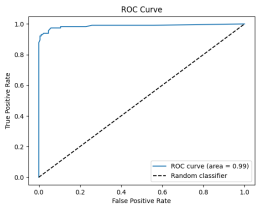

Fig. 6. ROC-Curve of SHAP-based Feature Reduction.

Fig. 7. ROC-curves from Feature Reduction

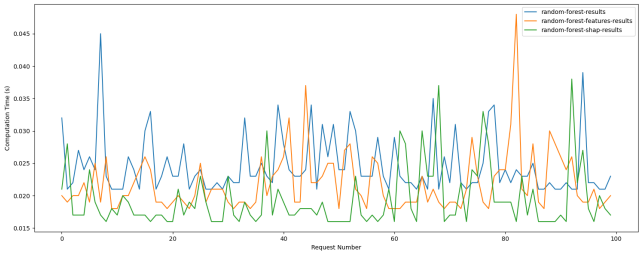

Fig. 8. Random Forests computation times comparison.

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 for both approaches 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.

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.

4.3 Quantization - Deep NN

For a 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 used to test Deep NNs is a Multilayered Perceptron (MLP).

Firstly in Figure 9 is displayed the confusion matrix of the default MLP model, for comparison purposes against the two forms of quantization of the MLP weights. 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.

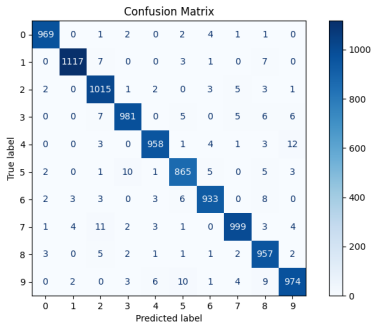

Fig. 9. Default MLP confusion matrix.

Figure 10 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).

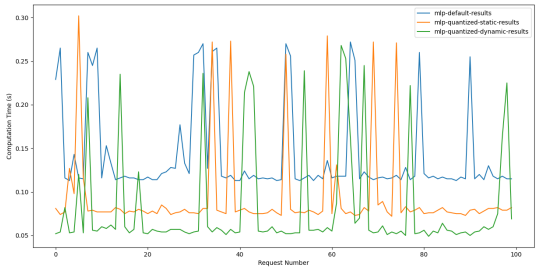

Fig. 10. MLP computation times comparison.

Figure 11 and Figure 12 show the resulting confusion matrices 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%.

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

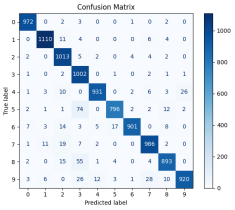

Fig. 11. Confusion Matrix of MLP with static quantization.

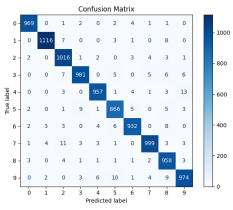

Fig. 12. Confusion Matrix of MLP with dynamic quantization.

Fig. 13. Confusion Matrices resulting of quantization.

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.

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, although static quantization offers better results, static could still be used to viable extents.

4.4 Surrogate Models

The surrogate model to use for Isolation Forests consists of a Kriging Partial Least Squares (KPLS) regression. It combines the Kriging Gaussian Process with Partial Least Squares. 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 14 depicts how the comparison of the original model with the final iteration of the surrogate model.

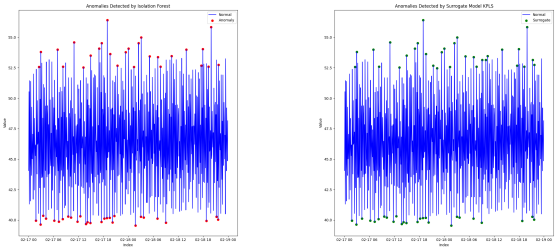

Fig. 14. Original model vs surrogate anomaly detection comparison.

Figure 15 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).

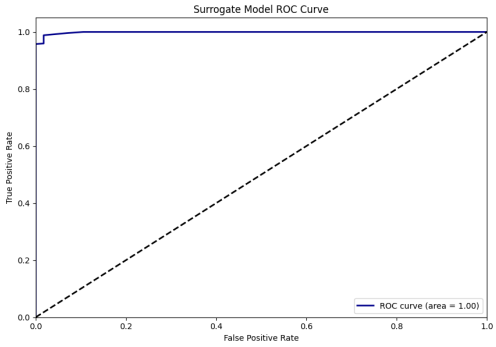

Fig. 15. Original model vs surrogate model comparison.

How the surrogate model leverages against the original Isolation Forest in terms of computation times is described in Figure 16. 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.

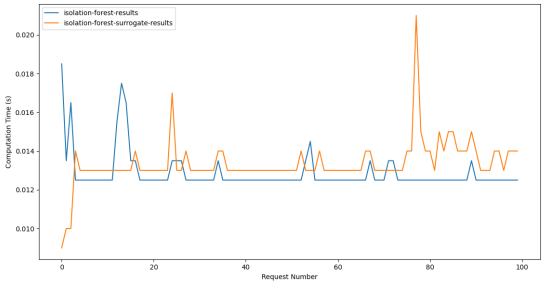

Fig. 16. Original model vs surrogate model comparison.

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. 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 effectiveness 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.

5 Conclusions

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.

During the work we present a systematic approach to determine if several techniques are fir to incorporate and analyse in this study, a distributed architecture on which the framework is developed and tested on, as well as explained the final cost-reduction techniques that were deemed as viable to study. We formulate a Cost Model as well as an algorithmic approach to evaluate each technique. Then finally, we tested the techniques in the framework and applied the cost model as well as the algorithms to ponder on its value on production systems.

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, adaptive execution [Esteves et al. 2018] and resource allocation [Simão and Veiga 2012] or even try ML oriented techniques like surrogate models, but with the original model as backup. 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.

Acknowledgments. To the supervisors of my Master Thesis, Prof. Luís Veiga and Pedro Patrício. To all my friends and family.

References

Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: Machine learning inference serving on serverless platforms with adaptive batching. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020). https://doi.org/10.1109/sc41405.2020.00073

Reza Alizadeh, Janet K Allen, and Farrokh Mistree. 2020. Managing computational complexity using surrogate models: a critical review. *Research in Engineering Design* 31, 3 (2020), 275–298.

Amazon. 2019. Debiasing AI using Amazon sagemaker. https://aws.amazon.com/sagemaker/ Acessed on 30 Nov 2023.

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. 2013. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28. PMLR, 199–207.

Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2023. Trends in AI Inference Energy Consumption: Beyond the performance-vs-parameter laws of Deep Learning. *Sustainable Computing: Informatics and Systems* 38 (2023), 100857. https://doi.org/10.1016/j.suscom.2023.100857

Sérgio Esteves, Helena Galhardas, and Luís Veiga. 2018. 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*, Paulo Ferreira and Liuba Shrira (Eds.). ACM, 239–252. https://doi.org/10.1145/3274808.3274827

Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring {HTTPS} adoption on the web. In *26th USENIX security symposium (USENIX security 17)*. 1323–1338.

Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. 1999. *Hypertext transfer protocol–HTTP/1.1.* Technical Report. W3C/MIT.

Eleonora Florian, Fabio Sgarbossa, and Ilenia Zennaro. 2021. Machine learning-based predictive maintenance: A cost-oriented model for implementation. *International Journal of Production Economics* 236 (2021), 108114. https://doi.org/10.1016/j.ijpe.2021.108114

Google. 2024. gRPC. https://grpc.io/ Accessed on 30 Aug 2024.

Yitao Hu, Rajrup Ghosh, and Ramesh Govindan. 2021. Scrooge, In Proceedings of the ACM Symposium on Cloud Computing. *Proceedings of the ACM Symposium on Cloud Computing*. https://doi.org/10.1145/3472883.3486993

Arya Iranmehr, Hamed Masnadi-Shirazi, and Nuno Vasconcelos. 2019. Cost-sensitive support vector machines. *Neurocomputing* 343 (2019), 50–64. https://doi.org/10.1016/j.neucom.2018.11.099

Grafana Labs. 2014. Grafana. https://grafana.com/docs/ Acessed on 5 Jan 2024.

Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. 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*. 1–15.

Kuan-Ming Lin and Chih-Jen Lin. 2003. A study on reduced support vector machines. *IEEE Transactions on Neural Networks* 14, 6 (2003), 1449–1459. https://doi.org/10.1109/TNN.2003.820828

Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, Venkatesh Babu Radhakrishnan, and Anirban Chakraborty. 2019. Zero-Shot Knowledge Distillation in Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 4743–4751.

Mary Phuong and Christoph Lampert. 2019. Towards Understanding Knowledge Distillation. In *Proceedings of the 36th International Conference on Machine Learning*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 5142–5151.

Cloud Native Prometheus. 2014. Prometheus - Monitoring System & Time Series Database. https://prometheus.io/ Acessed on 5 Jan 2024.

José Simão and Luís Veiga. 2012. QoE-JVM: An Adaptive and Resource-Aware Java Runtime for Cloud Computing. In *On the Move to Meaningful Internet Systems: OTM 2012*. Springer Berlin Heidelberg, Berlin, Heidelberg, 566–583.

Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2022. Enabling Cost-Effective, SLO-Aware Machine Learning Inference Serving on Public Cloud. *IEEE Transactions on Cloud Computing* 10, 3 (2022), 1765–1779. https://doi.org/10.1109/TCC.2020.3006751

Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).