



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE



End-to-End Anomaly Detection System in the CERN OpenStack Cloud Infrastructure

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE ENGINEERING - INGEGNERIA INFORMATICA

Author: **Stiven Metaj**

Student ID: 925266

Advisor: Prof. Giacomo Boracchi

Co-advisors: Domenico Giordano

Academic Year: 2021-22



*I wish there was a way to know you're in the good
old days before you've actually left them*

Abstract

The CERN OpenStack private data center offers cloud resources, services and tools to a scientific community of about 3,000 CERN users. In the infrastructure, about 14,000 virtual machines are deployed, covering several use cases, from web front-ends to databases and analytics platforms. Cloud service managers have to make sure that the desired computational power is delivered to all the users. To accomplish this task, spotting anomalous server machines in time is crucial. The previously adopted solution consists in monitoring the performance metrics of the machines using a threshold-based alarming system.

In this thesis, we present the new anomaly detection system that currently runs in the CERN cloud infrastructure. Given the mentioned multi-variate time series metrics, we run three different unsupervised machine learning models: Isolation Forest, LSTM-autoencoder and GRU-autoencoder. Then, using different ensemble strategies, the system automatically proposes the most anomalous servers of the previous day to the CERN cloud managers. We show the related end-to-end pipeline going from the data sources to the detected anomalies, the details of the architecture of the system, the pre-processing steps implemented, and the design choices regarding our solution.

Furthermore, we present a new labelled evaluation dataset related to the CERN cloud use case, and the results of our experiments with respect to it. Particularly, we show that the three adopted models, despite their very different nature, all achieve high performance in terms of AUC-ROC. Furthermore, we show that both the individual models and the ensemble strategies outperform the previous system in terms of true positive rate, for the given false positive rate required by the data center's operators. In addition, we compare the time performance of the models, and we show that the training is robust to the selection and size of the training set.

Keywords: Cloud Computing; Data Center; CERN; Anomaly Detection; Multi-Variate Time Series; Machine Learning; Isolation Forest; LSTM; GRU; autoencoder

Sommario

Il centro dati privato basato su OpenStack del CERN offre risorse, servizi e strumenti cloud a una comunità scientifica di circa 3.000 utenti. Nell'infrastruttura vengono implementate circa 14.000 macchine virtuali, che coprono diversi casi d'uso, dai front-end web ai database e alle piattaforme di analisi. I gestori dei servizi cloud devono assicurarsi che la potenza di calcolo desiderata venga fornita a tutti gli utenti. Per portare a termine questo compito, è fondamentale individuare in tempo i server anomali. La soluzione precedentemente adottata consiste nel monitorare le metriche dei server mediante un sistema di allarmi basati su soglie statiche.

In questa tesi, presentiamo il nuovo sistema di rilevamento delle anomalie attualmente in esecuzione nell'infrastruttura cloud del CERN. Date le metriche di serie temporali multivariate menzionate, eseguiamo tre diversi modelli di apprendimento automatico senza supervisione: Isolation Forest, LSTM-autoencoder e GRU-autoencoder. Quindi, utilizzando diverse strategie di insieme, il sistema propone automaticamente ai gestori cloud del CERN i server più anomali del giorno precedente. Mostriamo la relativa pipeline end-to-end che va dalle origini dei dati alle anomalie rilevate, i dettagli dell'architettura del sistema, le fasi di pre-processing implementate e le scelte progettuali relative alla nostra soluzione.

Inoltre, presentiamo un nuovo set di dati di valutazione etichettato relativo al nostro caso d'uso e i risultati degli esperimenti relativi ad esso. In particolare, dimostriamo che i tre modelli adottati, nonostante la loro diversa natura, hanno tutti prestazioni elevate in termini di AUC-ROC. Inoltre, mostriamo che sia i singoli modelli che le strategie di insieme superano l'attuale sistema utilizzato al CERN in termini di tasso di veri positivi, per il dato tasso di falsi positivi richiesto dagli operatori del centro dati. Inoltre, confrontiamo le prestazioni temporali dei modelli e mostriamo che il training è robusto rispetto alla selezione e alle dimensioni del training set.

Parole chiave: Computazione Cloud; Centro Dati; CERN; Anomaly Detection; Time Series Multi Variate; Machine Learning; Isolation Forest; LSTM; GRU; autoencoder

Acknowledgements

I would like to thank Prof. Giacomo Boracchi for presenting me the possibility to work on this project, supervising the scientific aspects of the thesis, and pushing me to achieve the desired goals. Thanks to my supervisor at CERN, Domenico Giordano. You gave me this opportunity in the first place, and you helped in the everyday work guiding me through the project during the last year.

I will be always grateful to both of you for allowing me to live this amazing experience, which improved tremendously my overall technical skills.

Thanks, then, to my parents. Despite the distance, you kept supporting me in the great, and especially, in the rough times during the last years of my master's. Thank you for all the sacrifices you made for my sake and for the encouragement you always gave me regarding my life choices. Thanks to you, coming back home is always a joyful event.

I want also to thank all the people who made this long journey more pleasant and less difficult. I will start thanking the guys from Florence who I firstly met at the Polytechnic. Thanks to you, moving to the new residence was less challenging, and studying together will be always a great memory. I owe you a lot. Then, thanks to all the new friends I met in Milan, you were like a second family. Thank you for all the good quality conversations in the common kitchen and for the support you gave me while I was moving to Geneva. In Geneva I also met amazing people. Thank you for improving my English, for the parties, for the nights spent studying together, for the trips, and why not, also for the dramas.

Finally, thanks to Antonin and to all the others colleagues I met at CERN. Due to restrictions we didn't have many possibilities for interactions, but I enjoyed all the external activities, all the coffee breaks and all the croissants we shared.

Contents

Abstract	iii
Sommario	v
Acknowledgements	vii
Contents	ix
List of Figures	xiii
List of Tables	xv
List of Algorithms	xv
1 Introduction	1
1.1 Context	1
1.2 Motivations	5
1.3 Methodology	7
1.3.1 Methodology: AD System	7
1.3.2 Methodology: Experiments	10
1.4 Contributions	11
1.5 Structure of the Thesis	12
2 State of The Art	15
2.1 Cloud Computing	15
2.1.1 Performance Metrics	18
2.2 Cloud Computing at CERN	18
2.2.1 OpenStack	19
2.2.2 Monitoring Metrics at CERN	20
2.3 Anomaly Detection	21

2.3.1	Anomaly Detection Techniques	24
2.4	Anomaly Detection on Time Series	28
2.4.1	Isolation Forest	28
2.4.2	Autoencoders	30
2.5	Related Work and Challenges	34
2.6	Summary	34
3	Proposed Solution	37
3.1	Problem Formulation	37
3.2	Time Series Pre-processing	39
3.3	Anomaly Detection Approach	41
3.4	Ensemble	43
3.5	Actors	43
3.6	Architecture	45
3.6.1	Pipeline	45
3.6.2	Orchestration and Scheduling	48
3.7	Monitoring Dashboards	53
4	Implementation Details	55
4.1	Anomalous Scenarios	55
4.1.1	Scenario 1: GitLab CI VM Slowdown	55
4.1.2	Scenario 2: Increasing Disk IO Time	57
4.2	Goals and Requirements	59
4.3	Background	60
4.4	Design Decisions	61
5	Evaluation Dataset	65
5.1	Lack of Public Datasets	65
5.2	Creation Process	66
5.3	Dataset Description	67
6	Experiments	69
6.1	Figures of Merit	69
6.1.1	Precision, Recall, F1-Score	69
6.1.2	Receiver Operating Characteristic and AUC-ROC	71
6.2	Experimental Setup	72
6.2.1	Parameters Selection	73
6.3	Results	74

6.3.1	Performance of the Proposed Solution	74
6.3.2	Comparison with Previous System	76
6.3.3	Minor Results	77
6.4	Discussion	81
7	Conclusions and Future Work	83
7.1	Outputs and Contributions	83
7.2	Future Work	84
	 Bibliography	 87
	 List of Symbols	 93
A	Appendix - Detailed Results	95
B	Appendix - Code Snippets	99
B.1	Utils	99
B.2	ETL Steps	102
B.3	ETL Pipelines	104
B.4	PyOD Wrapper	107
B.5	GRU-AutoEncoder	109
B.6	Airflow	110
B.7	Unit Tests	111

List of Figures

1.1	The resources available and used in the CERN cloud.	2
1.2	The difference between bare-metal servers and hypervisors.	4
1.3	The Iterative Development diagram.	9
2.1	The different levels of the cloud computing architecture.	17
2.2	The overview dashboard of the CERN OpenStack cloud.	19
2.3	The monitoring pipeline and the related technologies used at CERN. . . .	20
2.4	Examples of anomalies in a 2-dimensional hyperspace.	23
2.5	Examples of multi-class and one-class problems	25
2.6	Taxonomy of the main DL AD Techniques.	27
2.7	iTree Example	28
2.8	Architecture of an autoencoder	31
2.9	Comparison between autoencoders and PCA.	31
2.10	Structure of the LSTM cell with the related formulas.	32
2.11	Example of a standard LSTM-autoencoder architecture.	33
2.12	Structure of the GRU cell with the related formulas.	33
3.1	Statistics related to <i>disk IO Time</i> and <i>CPU load</i>	38
3.2	Heatmap representation of metric window.	41
3.3	The main actors of the CERN cloud.	45
3.4	The conceptual pipeline of our AD system.	46
3.5	The <i>Airflow</i> DAG we run.	49
3.6	The schedule of our AD pipeline.	50
3.7	Gantt chart related to the first run and the following ones.	52
3.8	Duration of the tasks of the AD pipeline.	52
3.9	The Grafana monitoring dashboard for ensemble results.	53
3.10	The Grafana monitoring dashboard for single results.	54
4.1	Scenario 1: list of VMs in the target hypervisor.	56
4.2	Scenario 1: monitoring dashboard of the target hypervisor.	56
4.3	Scenario 2: monitoring dashboard of the target hypervisor.	57

4.4	Scenario 2: starting phase of the anomaly.	58
4.5	Scenario 2: AD results	58
5.1	The first step of the dataset creation.	66
5.2	The second step of the dataset creation.	67
5.3	Heatmap of the labelled dataset.	68
5.4	Examples of labelled anomalies.	68
6.1	ROC curves examples.	71
6.2	SWAN details.	72
6.3	IFOR performance changing number of iTrees.	75
6.4	A visualization of the sliding window technique.	78
6.5	Increasing the amount of data and applying the sliding window.	79
6.6	Increasing both the amount of data and the number of dimensions.	79
6.7	Time performance of IFOR increasing iTrees	80
7.1	Our AD system in action on the CERN Cloud.	84
A.1	Loss function of LSTM-AE and GRU-AE during the training.	95
A.2	ROC curve and score distributions of the three models.	96
A.3	Old system dashboard.	97
A.4	AUC-ROC of the three models changing month for the training data.	97

List of Tables

2.1	The main Collectd metrics used in the CERN cloud infrastructure.	22
6.1	The confusion matrix of a binary problem.	69
6.2	The values of the main parameters of our system.	73
6.3	AUC-ROC comparison with 1 week of training.	74
6.4	AUC-ROC comparison changing LSTM and GRU parameters.	76
6.5	Comparison between our approach and the previous system	77
6.6	Training and Inference time of the three models.	81

List of Algorithms

2.1	$iForest(X, t, \theta)$	29
2.2	$iTree(X, e, l)$	29
2.3	$PathLength(x, T, e)$	30

1 | Introduction

Cloud computing is becoming the default paradigm to follow and to implement in many fields. The flexibility and the efficiency offered by it drive the choice of the companies that want to be competitive and able to scale applications in a dynamic way. Furthermore, for achieving this standard, there is the need for underlying hardware capable of supporting it and also the need of detecting and managing the failures that can happen. Indeed, cloud computing permits the building of more complex applications, which, on the other hand, makes it more probable to have system failures during the lifetime of the system itself.

These failures can propagate in the system with a cascade effect. For instance, a system that doesn't work properly for some hours can lead to a general dissatisfaction of the users or, even worse, to a decrease in the value of the company in the market. For this reason, it is always better to detect these errors as soon as possible in order to then decide which actions to take for solving the problem.

In this chapter we will introduce the context of our project, we will understand the possible scenarios, we will formulate in a formal way our problem, we will see the methodology and the contributions of the work of this thesis, and, finally, we will list the chapters of the thesis describing what we will show for each of them.

1.1. Context

CERN (Conseil Européenne pour la Recherche Nucléaire) is the European Organization for Nuclear Research, one of the world's largest centres for scientific research regarding particles and high Energy Physics. The main goal of the organization is to provide particle accelerators and other infrastructures needed for high-energy physics research. As a result, numerous experiments have been constructed at CERN through international collaborations. The main site at Meyrin, in the Geneva area, hosts a large computing facility, which is primarily used to store and analyze data from experiments, as well as simulate events. Researchers need remote access to these facilities, so the lab has

historically been a major wide area network hub.

To provide resources to the scientists and to accomplish its research goals in High Energy Physics, CERN relies on the computational power of its data center. The computing facility is structured as a private cloud, managed via components of the free open-source Infrastructure-as-a-Service platform OpenStack (better explained in the section 2.2.1).

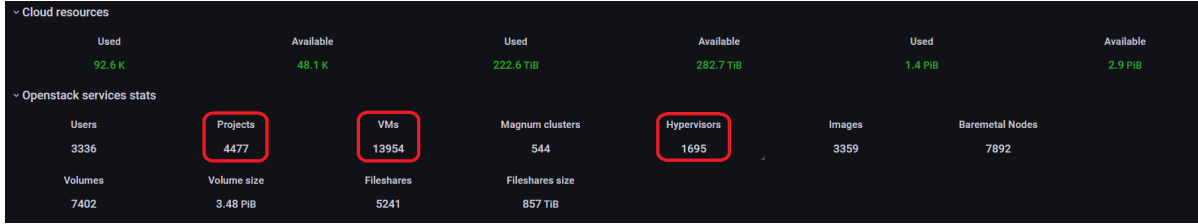


Figure 1.1: The resources available and used in the CERN cloud. We can see the number of projects, users, virtual machines, and hypervisors involved in the infrastructure.

The CERN personnel structure consists of different departments where each department is divided into groups, each group is divided into sections and in each section, there are multiple projects going on. For instance, the team who worked on this thesis project is part of the Resource Provisioning Service section, which is included in the Compute and Monitoring group, which is comprised of the Information Technology department (IT-CM-RPS). At the time of writing, the CERN OpenStack cloud contains about 10000 bare-metal servers where 2000 of them, configured as hypervisors, host about 14000 virtual machines, serving 4500 different projects, as shown in Figure 1.1. Each hypervisor, during its lifespan, produces several performance metrics related to its components. Those metrics are time series, and their values usually reflect the status of the hypervisor.

Definition 1.1.1 (Bare-metal server). A bare-metal server, also known as single-tenant physical servers or managed dedicated server, is a physical computer server that is used by one consumer, or tenant, only. Each bare-metal server offered by the infrastructure is a distinct physical piece of hardware that is a functional server on its own. They are not virtual servers running in multiple pieces of shared hardware.

Definition 1.1.2 (Hypervisor). A hypervisor, also known as a virtual machine monitor, or also called *server* in this thesis, is computer software, firmware or hardware that creates and runs virtual machines. A hypervisor allows one host computer to support multiple guest virtual machines by virtually sharing its resources, such as memory and processing.

In this document, we will call *bare-metal servers* those servers that don't use virtualization and that are not part of the cloud infrastructure we will monitor. On the contrary, we

will use *hypervisor* for those bare-metal servers with virtualization capabilities, which compose the cloud we will monitor and study.

Definition 1.1.3 (Virtual Machine). A Virtual Machine (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual “guest” machines run on a physical host machine (the *hypervisor*). Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host. For instance, this means that a MacOS VM and a Windows VM can run on the same physical server.

Others definitions follow. They will be useful for the next chapters of the thesis.

Definition 1.1.4 (Metric (or plugin)). As visible in Table 2.1, we refer to metrics, or plugins, when we talk about the performance metrics of the hypervisors. Those metrics are represented by time series, and they are related to the main component of the machines. The main known metrics are related to the disk, the CPU, the memory, and the network).

Definition 1.1.5 (Cloud managers). At CERN, we call cloud managers those employees who have a deep knowledge of the cloud infrastructure of the organization and are able to solve the issues detected by the users. They usually work also on new implementations regarding the cloud and, in particular, they are our main final users. Cloud managers will use our system, giving feedback to improve it or feedback to let us understand if the results are valid and if they can be used in the daily operations.

Definition 1.1.6 (Monitoring dashboards). The Monitoring dashboards are a set of information, plots, tables and links used for monitor the situation of the cloud machines. Usually they are implemented using *Grafana*, the main tool used by the central monitoring infrastructure. Many of the plots show the metrics related to a specific hypervisor or to a specific set of hypervisors, for instance, we can monitor statistics about a group of them.

We must highlight anyway that, during the last year, the number of servers in the cloud infrastructure changed drastically, as showed in Figure 2.2. The number of users and project is almost the same, but the infrastructure evolved and it was optimized involving less servers in general. As a consequence, misbehavior are more probable because of the less used resources, and for this reason, an AD system, running continuously on the CERN cloud, is even more decisive.

Furthermore, the data center and the cloud are used for high throughput computations related to the physics experiments and to the applications or services offered to the whole CERN personnel. In both cases, a misbehavior of the infrastructure can have an impact to the quality of service for the users. For this reason, it is fundamental to appropriately

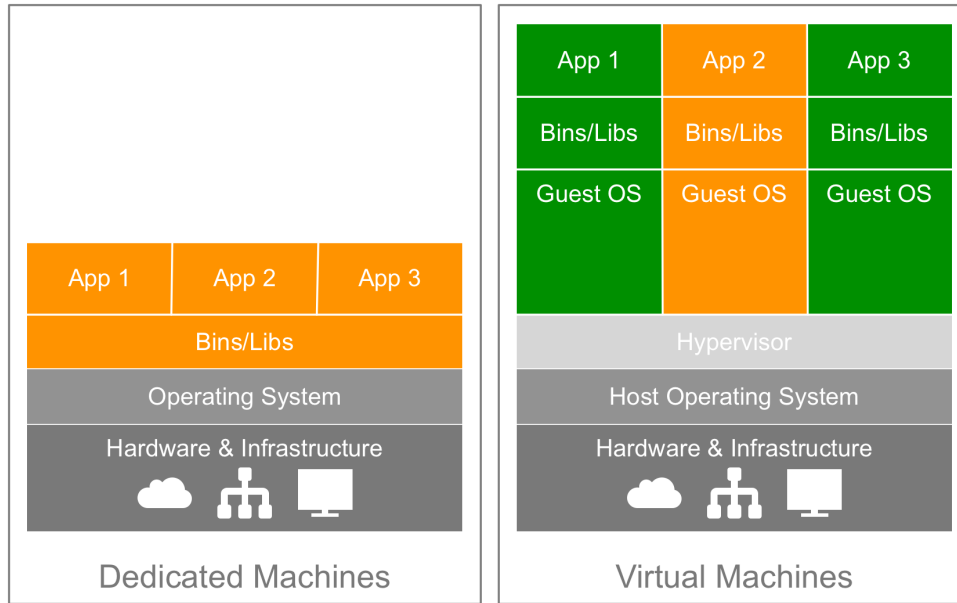


Figure 1.2: The difference between bare-metal servers, on the left, and hypervisors, on the right. In the first case, we have a single OS running different applications. In the second case, we have the same hardware and infrastructure, but, adopting the virtualization paradigm, we are able to have different guest operating systems running different applications independently.

monitor the cloud and to be sure that self-healing procedures are implemented when possible [1].

Regarding the monitoring aspect, the status of the cloud infrastructure is observed through multiple dashboards where is also possible to have alarms reporting high values to the cloud managers. In particular, the alarming systems used at CERN are mainly composed by threshold-based triggers. Those triggers usually capture only well-known problems, sometimes they notify false positives and finally, they tend to overwhelm the experts with unneeded notifications.

We find here the main reasons behind the start of our work. We need an alternative approach for detecting automatically possible anomalies in the cloud infrastructure, in order to avoid the negative aspects of the previous methods used for alerting and to benefit from all the data we are able to capture from the hypervisors. Our approach is based, indeed, on Machine Learning techniques. We want a system not driven by predefined rules, but by the data itself. Our new system, in particular, makes use of unsupervised methods because of the lack of an adequate number of labels. Moreover, having an unsupervised approach reduces the costs in general, doesn't need to have new

labels if the infrastructure evolves, and optimizes the system for future uses also for slightly different use cases.

The mentioned Machine Learning models we are considering are defined as AD algorithms (Section 2.3) and the time series that we get from the hypervisors' metrics will be our inputs (Section 2.4). In particular the models used by our system and considered in the experiments will be three, coming from two different families of algorithms. We will consider, from the family of traditional statistical models, the Isolation Forest, explained better in the section 2.4.1. On the other hand, from the family of Deep Learning models, we will consider the LSTM-autoencoder and the GRU-autoencoder, explained better in the section 2.4.2.

1.2. Motivations

Given the context of the project, it is also important to understand the specific target problem that we want to solve. We analyze here the scenario in which our system will work, and we highlight the motivations of our study.

The typical series of events we are focusing on is the following:

1. An hypervisor starts to have some kind of problems. The possible problems' natures are many. For instance, it is possible that a particular VM is consuming too many resources, or the disk can be in its degrading phase. The specific reason is not important for our case study, we only care about the detection of those anomalies.
2. After a not well-defined period of time, the problems of the hypervisor have a noticed impact on one or multiple users. Also here many scenarios can be true. It is possible that some experiments are producing wrong results, or that they are taking more time than usual. Maybe some experiments can not even be run because of some hardware error, or perhaps, a service breaks and the related users start to encounter issues.
3. After one or more of those episodes, the afflicted user will create a ticket in order to report the issue. These kinds of tickets will propagate until reaching one of the CERN cloud service managers.
4. The cloud manager who takes charge of the ticket will try to get to the source of the problem described by the user, finding also a way to solve the issue. Also here we have many different ways to solve these kinds of situations, but again, we are not really interested in this aspect.

5. After the changes are done by the cloud manager, the user is notified and the metrics of the problematic hypervisors are monitored to understand if the problem is finally solved.

As you may already noticed, all those steps require the coordination of many services and many people. Moreover, it takes a lot of time from the beginning of the anomaly to the final resolution of the issue, and in particular, this time is totally lost with respect to the hypothetical service down or to the experiment that couldn't continue.

The Anomaly Detection (AD) system proposed by the work of this thesis aims, indeed, to cut all those steps, trying to detect anomalies during the first step, before any service or any user is really affected by it. The keyword of this approach is prevention. Instead of solving reported issues, the goal is to prevent problems by checking if any anomaly is present in his early stages.

In the CERN cloud, a prevention system is already implemented, but it is not enough. Indeed, as mentioned in the previous section, the monitoring infrastructure used by the cloud managers offers a threshold-based alarming system able to alert and notify them in case a specific metric of a machine goes above a specific value. However, even if this method is the simplest possible one, and even if it is, after all, a good way to identify extreme anomalous cases, it has many weak points that motivate us to implement a new data-driven system for the AD purpose:

- The threshold-based alarms are related to single metrics. On the opposite, our system considers a multi-variate context where the combination of many metrics can identify an anomaly, especially in the early stages of the possible issue.
- The thresholds are static values that must be decided in advance. This is a wrong approach from many different points of view: different metrics need different thresholds, with the evolution of the infrastructure also the threshold should change accordingly, and if a specific metric of a hypervisor is high and very close to the threshold, but not above it, there is no notification.
- The described system is not able to give a ranking of the anomalies.
- The managers have to check every single metric plot to understand the situation and, moreover, a hypervisor can be duplicated in many of the visualization plots if more than one metric is above the thresholds.
- Last but not least, everything about the alerting system is decided by the managers of the cloud and it is only driven by the personal experience and not by the data and by the statistics of the cloud itself. With our AD System, on the other hand,

we can benefit from the potentiality of both those two aspects.

A different approach using a data-driven model can overcome those issues. The Machine and Deep Learning models that we will take into consideration give us the possibility to use a multi-variate representation of an undefined number of metrics characterizing the specific hypervisor. We have to decide anyway a specific threshold to be used for having a binary output, but we can also avoid this last step and have a ranking of the anomalies, being more flexible and helping the managers in the prevention of possible issues. Furthermore, the output of the system can be visualized in a single dashboard or even in a single plot, making it easier to check the results. Finally, the models don't really need specific values to be decided, they look at the statistics of the data and they decide the underlying logic to be applied following those and not expecting inputs from the managers.

It is clear that such a model can really help. It can bring a lot of value to all the data that the data center is able to generate, and it can be a game-changer regarding the prevention and the detection of anomalies.

1.3. Methodology

The solution for building a system able to perform an AD task may seem easy, but the complexity given by working with the CERN infrastructure, by the amount of available new data every day, and by the constraints about frameworks and services to be used, makes it more difficult than expected. For this reason, it is crucial to specify our methodology, in order to both understand the steps that led us to the final solution and follow the thesis Chapters.

We can divide the methodology into 2 main parts. The first is about our focus on building such a system. The second is about making experiments for finding the best solution regarding the system itself, and for giving theoretical results.

1.3.1. Methodology: AD System

The approach for building the AD system is based on different aspects, following features of Design Science and Software Engineering. We present some of them in the following list.

- **Encapsulation:** we want that every step of the system is enough encapsulated to work without the need of the other ones, despite those steps that are logically before it. Moreover, we want to be able to easily understand if a specific step is broken or

not. Indeed, an easy way to debug should always be at the center of a project that is expected to last months or years.

- **Versioning:** a versioning framework is needed for being able to have the code always updated, and for having a way that makes it possible to both find older versions of the code and add new features in a pragmatic process.
- **Testing:** we need an environment for testing purposes. Our system, while running in production, will interact with many different services and frameworks. In order to not mess up with those, it is important to have a way to test the new and the old features of the system.
- **CI/CD:** we don't want to just test the whole system, but it is also necessary to apply procedures for ensuring Continuous Integration and Continuous Delivery of the code. In order to do it, we need to have automatic tests in our versioning framework, being able to test single parts of our system.
- **Pre-Processing:** raw data can be not standardized or they can have any kind of issues at any moment in time. A data pre-processing step is needed in order to perform data cleaning, to choose a specific data format always used, and to be sure to always have the same configuration without unexpected behaviors.
- **Optimization:** we want to optimize the operations and the resources in order to not repeat multiple times the same kind of computations. For instance, if we have already done the pre-processing of some data and we need them again, we don't want to re-process the original data, but we want to use the already processed ones.
- **Pipelining:** we need a pipeline going from the extraction of the initial data to the publishing of the anomalous results. Nevertheless, this pipeline must be run continuously with the new data that are available from the CERN central monitoring infrastructure.
- **Documentation:** in parallel with the progress of the code development, it is essential to keep updated the documentation. It is useful for both ourselves and other people interested in the library or in our system in general.

Many procedures, techniques, and methods are used to guarantee those aspects: Software Prototyping, Iterative Development, briefly shown in Figure 1.3, Performance Evaluation, Comparative Studies, and Simulations. After all those iterative processes, we are able to have a final pipeline able to work in a containerized way, able to produce results about the anomalousness of the hypervisors in the cloud infrastructure, and able to run in a testing environment with the possibility of debugging.

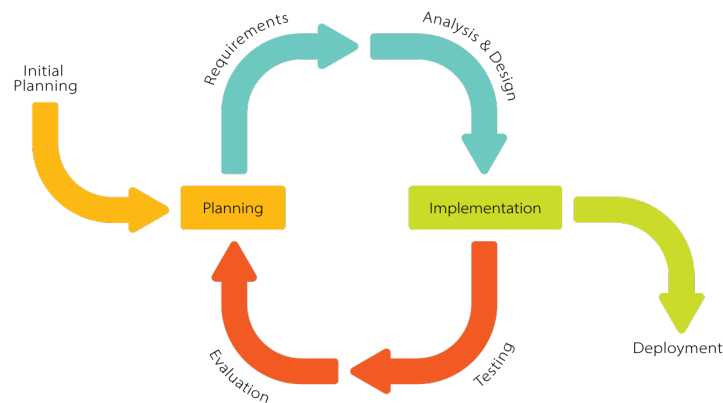


Figure 1.3: The Iterative Development diagram. Because of new required features or thanks to experts’ feedback, we had to re-think and re-implement many times the structure of our system.

To accomplish all the goals, many services, programming languages, frameworks, and libraries are used in a complementary manner. In the next chapters, we will focus more in detail on all those practical aspects, understanding better the services and the reasons that pushed us to use them. Here anyway, for summarizing those aspects and giving to the reader some hints about how we reached the final solution, we will list some of the tools we use:

- We use *GitLab* for the versioning and for the CI/CD jobs.
- For the containerization we use *Docker*. The Docker images are many and different depending on the purpose of the container. For instance, the Docker images used for connecting to the CERN services usually don’t change, while the one containing our library changes at every new push on *Gitlab*.
- In order to get the performance metrics, the *Collectd* service is used. It takes the time series from the hypervisors and saves the data using *HDFS*, the Hadoop Distributed File System.
- From *HDFS* we are able to process the data thanks to *Apache Spark*.
- The machine and deep learning models are implemented in Python using many libraries including *Sklearn*, *Pandas*, *PyOD*, and *TensorFlow*.
- The results are published from *Fluentd* to *ElasticSearch* with a logging mechanism.
- From *ElasticSearch* we visualize the final dashboard on Grafana or on *Kibana*, from where the cloud managers can consult them and send us the feedback.

- All the pipeline process is orchestrated and scheduled with *Apache Airflow*.
- The system runs in a virtualized environment created using the CERN cloud Openstack service itself. For instance, the machine on which our pipeline run is between the machines inspected by our system, being part of the cloud infrastructure under study.

1.3.2. Methodology: Experiments

In order to make experiments we used many of the techniques, tools, and procedures we saw in the methodology of the system. However, it is important to clarify some aspects that were mainly driven by the need for those experiments.

The first important distinction is the use of an interactive approach for writing the code. For the experiments is often crucial to be able to modify the code and the functions in the easiest possible way. Clearly, especially in terms of production, having a library with a versioning framework behind it is necessary, but still, it is not the best option in terms of flexibility. For those reasons, we decided to use a Python Notebook environment for all the work concerning experiments. In particular, we used the SWAN (Service for Web-based ANalysis)¹ framework: it is a CERN-based service and it allows interaction with the other services we are interested in. For instance, we want to connect to the CERN Spark cluster for the processing of the data, or we want to connect to the CERN *EOS* Storage framework in order to have compatibility with the operational system.

SWAN is used also for other purposes:

- to show to other colleagues, teams or departments the work done for this project,
- to make it possible for others to interact with our library or to consult our results, and
- to debug our library and to develop new features.

Secondly, we don't need only a tool to launch our experiments, but, most importantly, we need a labelled dataset for evaluating the different models or approaches we want to test. This is a crucial point, and moreover, the lack of annotated datasets related to our specific use case makes this even more important and time-consuming. The final decision is to create our own dataset using the CERN original data, with the constraint of reducing to the minimum the bias of the labels.

There is nothing more to add to the methodology related to the experiments. Despite

¹<https://swan.web.cern.ch/swan/>

the use of SWAN and the new dataset, we used the same other tools implemented for the system, with the only difference that our experiments are driven by a systematic literature review, statistical hypothesis formulations, and mathematical modeling following those hypotheses.

1.4. Contributions

During the project we were able to finalize many outputs that can be useful, in particular at CERN, for other users, other future students, and other teams that aim to solve similar problems. Moreover, we brought also contributions related to the performance of three AD models with respect to a unique cloud computing case study as the CERN one is. We also bring results related, more in general, to which preprocessing steps are better in a multi-variate context for this kind of objective.

All the contributions of this work, and especially all the details about them, will be clearer going ahead with the reading. However, here we want to give a comprehensive list of the goals that we achieved, in order to allow the reader to know what the project's accomplished intentions and objectives are:

- **AD library**²: before anything else, in whatever project, a library to be used in systems and experiments is necessary. This was also one of the main contributions of the previous work [2]. We started from it, improving the potentialities offered by the functions, generalizing them, and fixing all the bugs we found. The library is also used by other teams at CERN and by other people in different contexts. For instance, in the University of Victoria, Canada³, they adapted our library for their use case.

One of the most important aspects regarding the code is the ability to use specific parameters for the specific use of the functions. During the software engineering phase, the focus was always the minimization of the hard-coded parameters, in order to make it possible to use the library for other case studies. Furthermore, it is important to highlight that this library includes many functions related to the data science field. For instance, we have functions for data processing or data inspection, for including personalized Machine Learning models in the core of the system, for specifying which parameters and fields are needed in the results publishing, and for testing purposes.

²<https://gitlab.cern.ch/cloud-infrastructure/data-analytics/-/tree/qa-v0.4/>

³<http://heprcdocs.phys.uvic.ca/reports/kamel-wtr-2021.pdf>

- **Daily operational AD system:** after the creation of a modular, scalable and parametric library, the contributions of the project include the pipeline that we run automatically every day. The pipeline is in charge of computing the anomalous scores of the hypervisors included in the cloud infrastructure under study. This system is not only able to compute those results, but it optimizes the resources consumed, and it also publishes the scores using the same CERN monitoring framework used by all the users and managers involved in the cloud operations of the organization.
- **Evaluation dataset:** as already said, in order to make experiments, we need to create a labelled dataset. This is one of the main contributions because it doesn't only make possible all the theoretical work of the thesis, but it also sets the stage for future experiments or future dataset extensions. Furthermore, this dataset can be anonymized and published for other external experiments, or for public challenges following, for instance, the Kaggle⁴ format. Its importance, given the lack of labelled datasets in contexts like ours, actually increases. There are not many datasets dealing with multi-variate performance metrics related to cloud computing machines providing general usage resources. These kinds of data centers are often owned by big tech companies, and usually, they don't share data related to their servers' metrics.
- **Performance results and comparison of the systems:** the last contribution is related to the experiments we run on the evaluation dataset. We show empirical and quantitative results, highlighting the improvements that the new system brings to the CERN cloud use case. In particular, we present the pros and cons of the models that we run in the system, emphasizing the performance with respect to complexity, needed time for train or inference, and efficiency in finding anomalies. In addition, we benchmark the proposed models against the previous CERN alerting system, showing that both the individual models and the ensemble strategies outperform it in terms of True Positive Rate, for the given False Positive Rate required by the data center's operators.

1.5. Structure of the Thesis

As last part of the introduction let's introduce the next chapters mentioning briefly the topics included in each of them:

⁴<https://www.kaggle.com/>

- **Chapter 2, State of The Art:** here we will mention the major topics related to our thesis. In particular, we talk about cloud computing and how it is implemented in the CERN context, about the AD task and how it is implemented in the multi-variate time series context, and about the details of the specific models we will inspect and investigate.
- **Chapter 3, Proposed Solution:** here we will present our solution, formulate the problem, and show details about the data processing we need to perform and about the architecture of the daily operational system.
- **Chapter 4, Implementation Details:** in this chapter, we will go deeper into the design decisions of our solution. While in the previous chapter we mention the main points regarding our system, here we see some detailed scenarios where our system is needed, some background information about the project, the requirements of the system, and the related design choices.
- **Chapter 5, Evaluation Dataset:** after describing everything about our solution, we present the evaluation dataset that we created for running the experiments. In particular, we focus on the rules we used to have unbiased labels.
- **Chapter 6, Experiments:** here we will present and discuss our experiments and the related results. Moreover, we will introduce the metrics we use to assert the performance of the model and the parameters we decided to use for both the system and the experiments.
- **Chapter 7, Conclusions and Future Work:** finally, in this chapter, we will summarize the work of the thesis with a conclusion and we will talk about possible future work related to our system.
- **Appendix A, Detailed Results:** the first appendix is about the figures related to the results that can not be included in the main text because of their dimensions. Moreover, they are not fundamental to get the main messages we want to pass in the discussion of the results. However, we include them in this appendix for completeness.
- **Appendix B, Code Snippets:** the second appendix shows some snippets related to the code of our library.

2 | State of The Art

We approach the work of this thesis presenting the state of the art of the different entities composing our problem. In particular, we will go through the concepts of cloud computing and Anomaly Detection (AD), focusing on the aspects more important for us. Moreover, we want to learn more about the input data we will work with, about the expectations in a cloud computing environment, and about the most used unsupervised AD models in multi-variate settings.

The majority of the information we group here comes from research papers and studies about these topics. For instance, we will get most of the information about cloud computing from [3–6] and most of the details about different unsupervised techniques to perform Anomaly Detection from [7–10].

2.1. Cloud Computing

Thanks to the fast development of processing technologies, storage capabilities and Internet connectivity, we can rely on cheaper computing resources with better performance and in general more available than before. The result of this transformation is the transition to a new standard called cloud computing, in which resources like storage, memory or computational power are provided through the Internet and the users can utilize them as general utilities requesting or releasing them in an easy way.

The advantages for the end-users are many, and the main important ones are the following:

- **Higher scalability:** the cloud service can be expanded easily, and it is possible to be more transparent from the usability point of view.
- **Easier access:** the hosts in the cloud infrastructure are generally web-based and this means they are easily accessible through different devices with the only necessity of an Internet connection.
- **Reduction of risks:** the user doesn't have to think about possible hardware failures or about any possible problem related to the cloud infrastructure.

We can also define cloud computing as referring to [11]: *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Moreover, we can associate many technologies already known and used before to this new standard:

- **Grid computing:** the paradigm that consists in coordinating the resources in order to achieve a specific computational goal. Cloud computing is similar to this, but, to provide dynamic and shared resources, we have one more step by using virtualization at the hardware and at the application levels.
- **Virtualization:** the technology that allows to abstract the majority of the details about physical hardware and that provides the possibility of using virtualized resources giving more flexibility to the user applications. A virtualized server is called a Virtual Machine (VM) and usually, there are many different VMs running in single physical hardware called Host or Hypervisor.
- **Autonomic computing:** the paradigm that aims at having self-managing computing systems, capable of reacting possibly without any human intervention reducing the management complexity of the system.

Cloud computing shares some features with those technologies, but it also offers particular further benefits. They can be summarized with the following aspects:

- **Shared resource pooling:** the infrastructure offers a pool of dynamically assigned computing resources. This kind of assignment capability allows flexibility to the provider and to the end-user, managing the resource usage in an optimized way.
- **Geo distribution and ubiquitous network access:** as already mentioned, access to cloud infrastructures is guaranteed by the use of the Internet as a service delivery network. Moreover, cloud computing is also based on the fact that the infrastructure is distributed and that it consists of multiple data centers located in different locations around the whole globe.
- **Dynamic resource provisioning:** cloud computing differs from the traditional computing model also for the capability of providing and releasing computational resources on the fly. In the previous paradigms, resources were provided according to peak demand.
- **Self organizing:** the management of resource consumption follows the real-time

needs of the users. This yields high agility enabling a fast response to changes in service demand.

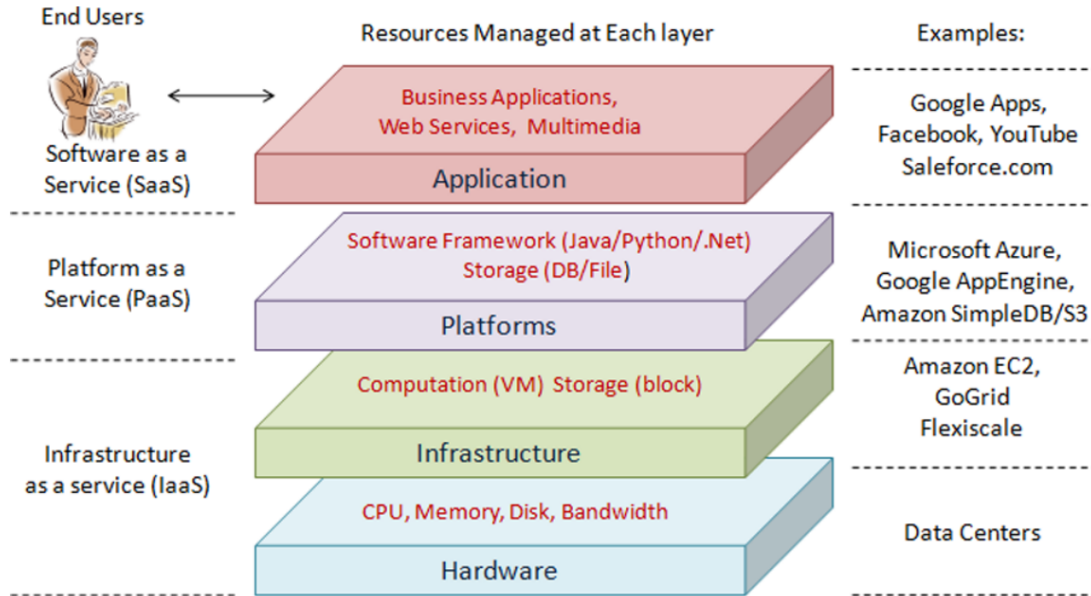


Figure 2.1: The different levels of the cloud computing architecture [3].

Finally, we can also visualize the architecture of cloud computing by looking at the different layers of it in Figure 2.1. We can identify 4 different layers. Going from the lowest to the highest level we have:

- **Hardware layer:** managing physical resources including physical server, routing devices, power and cooling systems. We can refer to this layer talking about the data center, considering a set of thousands of servers organized in racks and connected between each other through switches and routers.
- **Infrastructure layer:** creating a pool of computing and storage resources using virtualization. Essential layer that provides key features thanks to the partitioning of physical resources.
- **Platform layer:** handling operating systems and frameworks on top of the infrastructure Layer. Usually, there is API support for the implementation of the different aspects needed by the running applications.
- **Application layer:** running the actual applications. They have more features, better performance and flexibility thanks to the decoupling of the lower layers and to the modularity properties they have.

2.1.1. Performance Metrics

Going deeper into the topic of cloud computing, it is important to understand which metrics are involved. After all, they will be, in one way or the other, the inputs of our ML models and it is critical to know them. In [5] we have a complete list of all the metrics we can have, depending on the different paradigms we are interested in. On the other hand, we can be interested in evaluating commercial cloud services. In [6] we can find all the metrics for that purpose.

It probably doesn't make sense to list every possible metric, but it is fundamental to mention some of them in order to understand the next chapters, especially the ones related to the Monitoring of cloud services:

- **Resource utilization:** the percentage of resources (CPU, Memory, Bandwidth, ...) needed for the current workload.
- **Resource load:** the number of tasks waiting in the queue for utilizing the CPU.
- **Maximum running resource:** the maximum number of resources used in a specific setting.
- **Response and delay time:** numbers defined by the difference between the time of a sent request and the time of the processing of it.
- **Network congestion:** the incoming and outgoing network load, usually in terms of requests per second.
- **SLA violation:** the percentage of tasks having a delay time above a certain threshold.

Let's remember that this list is just a generalization and that, as we will see later on, each general aspect we are listing here is usually translated in different multiple low-level metrics. Those, defined by the time series, will be used as inputs.

2.2. Cloud Computing at CERN

We saw what we mean by talking about cloud computing and we listed general properties, architecture aspects and metrics which evaluate the performance. However, the complexity of the topic is also given by the fact that every use case differs from the others. Different companies or organizations implement different solutions to follow the cloud computing paradigm using various technologies. cloud services can be private, public or hybrid, it is possible to use commercial services or to build our own cloud and moreover, and, for

every segment of the service, there are many available technologies with their own pros and cons. Being this project focused on the CERN use case, it is important to understand how cloud computing is implemented, which technologies and frameworks are used, and which features or limitations we have.

In [12] we can find, for example, information about the history of the cloud infrastructure at CERN. In particular, the paper, published in 2015, goes deeper in the details on which experiments the cloud team did, and on which results they had comparing different possible solutions for supporting the ATLAS experiment from a resource provisioning point of view. Anyway, it is probably better to focus on more recent information about the cloud computing at CERN and especially to review the main aspects described in [13, 14].

2.2.1. OpenStack

Nowadays, at CERN, the selected cloud software to be adopted is OpenStack¹, an open-source cloud computing infrastructure software. It is an IaaS (Infrastructure as a Service) project and it is one of the most used in the field by many major companies. Firstly, the resources offered from the cloud team were mainly VMs and related attached volumes. Thanks to the improvements to the availability and to the increasing number of provided features, the cloud infrastructure is able now to offer also services like containers or network components.



Figure 2.2: The overview dashboard of the CERN OpenStack cloud. It is possible to understand how many physical and VMs are in the data center and the amount of data processed by the cloud computing infrastructure.

¹<https://www.openstack.org/>

The servers run CentOS² 7 and, after an extension in 2013, they are distributed into two different data centers located in Meyrin (Geneva, Switzerland) and Wigner (Budapest, Hungary). For management purposes, they are grouped into cells and then aggregated in a more scalable architecture. Thanks to this structure, it is possible to operate several cells, and it is easier to quickly grow the infrastructure with the changing demand of computing power. These machines are used for many purposes at CERN, some of the main use-cases are:

- workloads related to various experiments (LHC, LHCb, ATLAS, CMS [15–17]),
- schedule of the CERN Batch nodes [18],
- Continuous Integration and Development of the different teams' projects, and
- personal VMs for test purposes.

2.2.2. Monitoring Metrics at CERN

Once again, it can be useful to linger on the monitoring metrics. In particular, we want to highlight how they are managed in the CERN cloud infrastructure and which ones are used. In order to monitor the cloud, many services and technologies are used. In [19] we can find insights about them and about the whole pipeline going from the data sources to the metrics visualization, shown also in Figure 2.3. We are especially interested in understanding which is the technology that permits us to get those metrics, and we want to be aware of what each plugin is about.

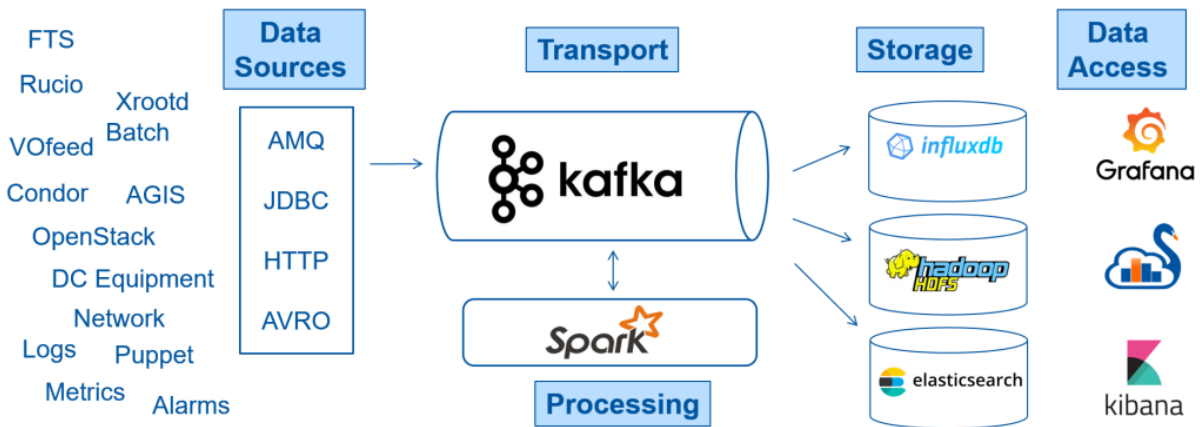


Figure 2.3: The monitoring pipeline and the related technologies used at CERN [19].

The first step of the monitoring pipeline is to provide a standard and solid data flow of

²https://hub.docker.com/_/centos/

those metrics. This task is accomplished by *Collectd*³. It is installed on all the nodes, and it gathers and sends all the metrics (defined in 1.1.4) to the central infrastructure, in order to permit all the possible next steps of the pipeline. Furthermore, it is used to collect IT services logs, generating almost 1 TB of data per day.

The collection of the metrics time-series allows the IT service managers to perform also additional operations like:

- configuring more metrics and developing new specific ones,
- building service dashboards to provide real-time indicators to be analyzed, and
- creating custom alarms in order get notified if something is wrong.

Finally, the list of the main plugins used in the CERN cloud infrastructure can be visualized in the Table 2.1. The complete list of all the possible plugins collected from *Collectd* can be found here⁴, but we want to focus our attention to the most important ones with respect to our use case.

2.3. Anomaly Detection

The next main topic of the thesis, after cloud computing, is, of course, Anomaly Detection. Indeed, if cloud computing is the main context of our use case, the detection of anomalies is what we are doing from the practical point of view, and it is fundamental to understand all the possibilities that there exist to comprehend how this task can be tackled. In the next paragraph, we will refer mostly to [7–10].

Anomaly Detection refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often referred to as anomalies, outliers, or exceptions depending on the different application domains. For instance, anomalies and outliers are the two terms commonly used in our context.

The task of detecting anomalies finds use in many applications such as fraud detection in financial settings, insurances, fault detection in safety-critical systems or military surveillance [7]. Anyway, before going deeper into the detection mechanism, it is essential to understand in the first place what are these anomalies. The definition of anomaly is often subjective, but, at the same time, it affects all the project.

Given a well-defined notion of normal behavior, anomalies are patterns in data that do not follow that notion. Those anomalies can be there for several reasons including

³<https://collectd.org/>

⁴<https://wiki.opnfv.org/display/fastpath/Collectd+Metrics+and+Events>

Plugin	Type	Type Instance	Description
CPU	percent	idle	Time CPU spends idle.
	percent	interrupt	Time the CPU has spent servicing interrupts
	percent	system	Time that the CPU spent running the kernel.
	percent	wait	The time the CPU spends idle while waiting for an I/O operation to complete.
	percent	user	Time CPU spends running un-niced user space processes.
Interface	if_dropped	in/out	The total number of received/transmitted dropped packets.
	if_octets	in/out	The total number of received/transmitted bytes.
Memory	memory	buffered	The amount, in kibibytes, of temporary storage for raw disk blocks.
	memory	cached	The amount of physical RAM, in kibibytes, left unused by the system.
	memory	free	The amount of physical RAM, in kibibytes, left unused by the system.
	memory	total	Total amount of usable RAM, in kibibytes, which is physical RAM minus a number of reserved bits and the kernel binary code.
Disk	disk_io_time	io_time	Time spent doing I/Os (ms). You can treat this metric as a device load percentage (Value of 1 sec time spent matches 100% of load).
	disk_ops	read/write	The number of read/write operations issued to the disk.
	pending_ops	-	The queue size of pending I/O operations.
Switches	switches	-	Number of context switches done by the operating system.
Load	load	shortterm	Load average figures giving the number of jobs in the run queue or waiting for disk I/O (averaged over 1 Minute).
	load	longterm	Load average figures giving the number of jobs in the run queue or waiting for disk I/O (averaged over 15 Minutes).

Table 2.1: The main Collectd metrics used in the CERN cloud infrastructure. The metrics in bold will be the ones used for our system.

malicious activities or spontaneous failures. In Figure 2.4 we can find an example in two dimensions. In particular, we can notice that the majority of the data is distributed in two bigger clusters (N_1 and N_2), while, on the contrary, some of the data points are more distant from the others and not inside those 2 groups (O_1 , O_2 , O_3).

Here comes the subjective aspect. O_1 , O_2 and O_3 can all be defined as anomalies, but it is possible, for instance, considering the points in O_3 not anomalous, because a new cluster is forming there. Another case consists in considering O_2 not anomalous because not so much distant from the main cluster.

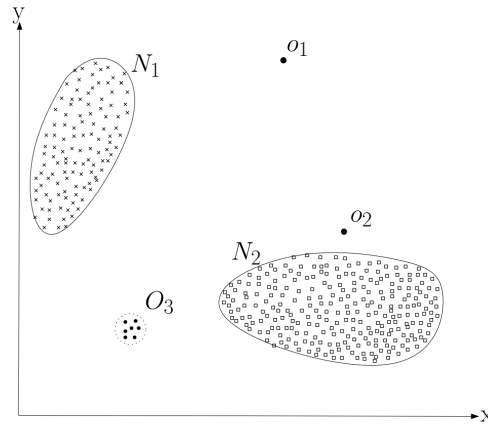


Figure 2.4: Various examples of anomalies in a 2-dimensional hyperspace^[7].

The definition of what is an anomaly is of primary importance to decide the methods and algorithms we can use for the AD task. Generalizing this concept, there are several kinds of possible anomalies that we might want to detect:

- **Point anomalies:** the simplest type of anomaly, when we can consider anomalous an individual data instance.
- **Contextual anomalies:** an anomaly given by a specific context, where the context can be induced from the structure of the dataset. For instance, two data points with the same attributes' values can differ from the anomaly point of view because of their different contexts.
- **Collective anomalies:** an anomaly given by a set of data points' values. The single data point value, in this case, doesn't matter at all. Only if other correlated points have certain values, we define that group of data points as anomalous.

Moreover we have also different type of AD depending on the labels availability of our dataset:

- **Supervised AD:** when there are enough labels for training a model. There are two main issues in this case. Firstly, the anomalous labels will always be far fewer with respect to the normal ones, having all the problems that come with unbalanced datasets. Secondly, in real applications and use cases, it is very difficult to label anomalies with great precision. In general anyway, once you have good labels for the anomalies, there is no need to use special models, and usually, it is “just” about building a predictive model.
- **Semi-Supervised AD:** when there are enough labels of normal data. The strategy, in this case, is to build a model learning only from the normal data and to use such a model to discover anomalies in the test data.
- **Unsupervised AD:** when there are no labels. This is the most common case because of the general difficulty of collecting good labels about anomalies. In addition, adding to the training dataset some of the anomalies from the test data, also semi-supervised techniques can be used in the same way. There is an important hypothesis anyway to consider. In the case of Unsupervised AD, we assume that normal instances are far more frequent than anomalies in the data. If this is not true, there will be many false positives.

2.3.1. Anomaly Detection Techniques

After the understanding of the general purposes of the AD task, we should focus on the various categories of techniques that it is possible to apply and implement, looking also for their pros and cons.

Classification Based Techniques

Classification means that there is a model capable of learning from a labelled dataset and of distinguishing from normal and anomalous data in the testing phase. In particular, we have multi-class techniques, in the case that the given problem has many normal classes, or one-class techniques, in the case the given problem has only one normal class. We can visualize the difference in Figure 2.5.

In this category, we find many possible models. Some of them are Neural Networks, Bayesian Networks, or Support Vector Machines. Usually, the testing phase of these algorithms is very fast and, using the right dataset, the models can be very efficient in distinguishing anomalies and normal data. On the other hand, it is not easy to have a good dataset, and moreover, in the testing phase, we get in output a specific label, while

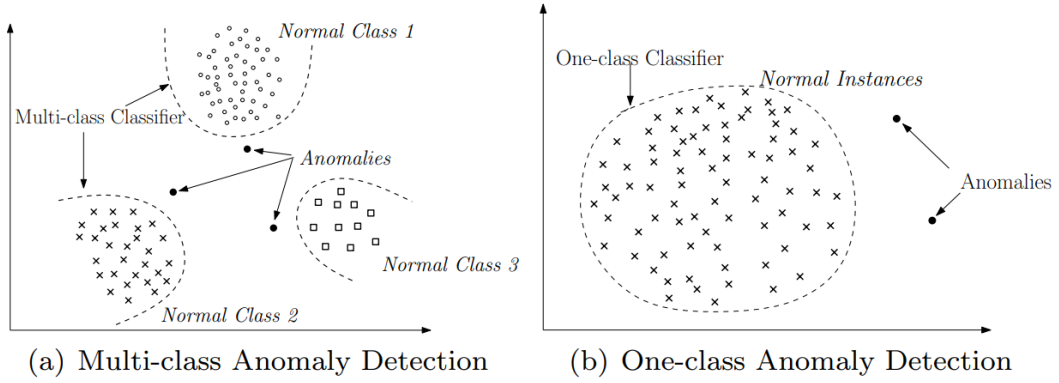


Figure 2.5: Examples of multi-class and one-class problems in a 2-dimensional hyperspace^[7].

sometimes we are interested in values indicating the anomalousness probability.

Nearest Neighbors Based Techniques

In this case, the assumption is that normal cases occur in dense neighborhoods while anomalies are distant from these groups. Surely, in order to use these algorithms, we have to define a distance, usually positive-definite and symmetric, and we have to implement different methodologies with respect to continuous and categorical attributes or to multi-variate instances. The algorithms that use this type of technique can be divided into two main groups: the ones using the distance of a data instance to its k_{th} nearest neighbor, and the ones computing the relative density of each data instance.

This class of techniques doesn't assume anything about the generative distribution of the data points, they can perform better in the case of semi-supervised adaptation and it is easier to use different data types if it is possible to define a distance. However, these models don't work properly if the normal data points don't have close normal neighbors, or if the distance is not well-defined with respect to the problem. Another issue is related to computational complexity. Indeed, computing the distance between each test instance and all the other instances belonging to either the test data itself or to the training data, leads to a $\mathcal{O}(N^2)$ complexity.

Clustering Based Techniques

With clustering techniques, it is possible to group similar data in specific clusters, usually without the need of labels. There are different clustering algorithms for different use cases, depending on the assumptions on the data distribution. There are three main potential assumptions we can find:

- Normal data instances belong to a cluster in the data, while anomalies either do not belong to any cluster.
- Normal data instances are close to their closest cluster's centroid, while anomalies are far from theirs.
- Normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters.

Given a specific assumption, we can use specific algorithms for that case. These models can work in an unsupervised manner, they can adapt for different data types and the computational complexity in the testing phase is low, because we always assume to have a low number of clusters. Unfortunately, we have also several drawbacks including a high dependency on the underlying clustering algorithm, no specific AD optimization, a big training phase complexity, reaching $\mathcal{O}(N^2d)$ in some cases, misbehavior in case the anomalies form clusters themselves.

Pure Statistical Techniques

In this category, we find all those techniques and strategies using the underlying assumption that normal data instances occur in higher probability regions of a stochastic model, while anomalies occur in lower probability regions of the same model. The idea is almost always based on fitting a model in order to learn the normal data behavior and then, to see in the test phase how good is the model to predict. For instance, the hardest is the prediction, the more probable is for the specific data point to be an anomaly. This is the case where we find the greatest number of different possible models. In particular, we have:

- **Parametric techniques:** here the normal data are assumed to be generated by a parametric distribution with parameters Θ and probability density $f(x, \Theta)$, where x is an observation. The anomaly score of a test instance x_i is the inverse of $f(x_i, \Theta_m)$, where Θ_m are the parameters estimated from the model given the data. Here we find Gaussian-Based models, Regression-Based models, and models using a mix of different Parametric Distributions.
- **Non-parametric techniques:** in this case, the model is not defined a priori. We only use the data are used and fewer assumptions are made. In this category, we find Histogram-Based and Kernel-Based models.

The advantages of statistical techniques are the fact that they have a well-defined confidence interval for the predictions, they are generally accurate if the distribution hypothesis

is valid, and they can be used for unsupervised techniques. On the negative side, there are the assumptions themselves, the difficulty on choosing the right hypothesis test statistics to be used, and the fact that some of the models don't perform well in multi-variate settings.

Deep Learning Techniques

In the last years, many Deep Learning (DL) models are being developed, studied and improved. Such models found also place in the AD task, where they are usually promising with respect to the normal ML models because of the ability, for instance, to reduce false positives, to find temporal dependencies between different attributes, or to not be infected from the possible noise of the test instances.

In Figure 2.6 the major techniques of this kind are shown. We can also identify, starting from the general idea of DL methods, three different more precise sub-categories: DL for feature extraction, Learning feature representations of normality and end-to-end anomaly score learning [10].

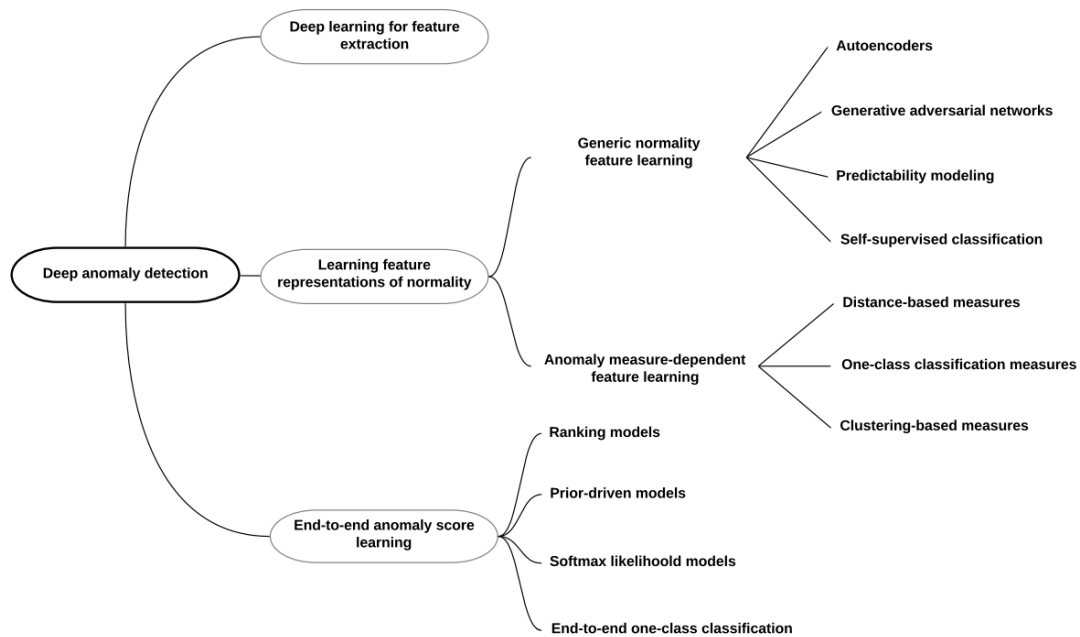


Figure 2.6: Taxonomy of the main DL AD Techniques^[10].

2.4. Anomaly Detection on Time Series

After an overall look at the general AD task, it is also important to focus on the specific use case of time series. Indeed, at the CERN data center, almost all the monitoring data received can be considered time-series and therefore, we must go through those AD models' adaptations that make it possible to use these kinds of input for accomplishing our desired goal.

In [20–23] it is possible to find a lot of information about the topic. In particular, we find details about dealing with discrete sequences where multiple problem formulations could be valid, building new proposals to improve speed and accuracy, using autoencoders or dealing with IoT time series. The strategies to handle time series are many and they all depend on the specific use case.

In our case, we focus on studying two main kinds of algorithms coming from two different worlds of AI. From the family of “traditional” ML models, we will focus on Isolation Forest [24], also called iForest or, even shorter, IFOR. From the family of DL models, we will focus on autoencoders. In particular, the autoencoders that use LSTM [25] and GRU [26] units. In the following subsections, we analyze these models more in-depth and, in particular, how they work in the case of time series inputs.

2.4.1. Isolation Forest

The idea at the basis of this model is that a tree structure can be constructed effectively to isolate every single instance, having isolated anomalies closer to the root of the tree and normal points, being probable part of a cluster, more distant from it (Figure 2.7).

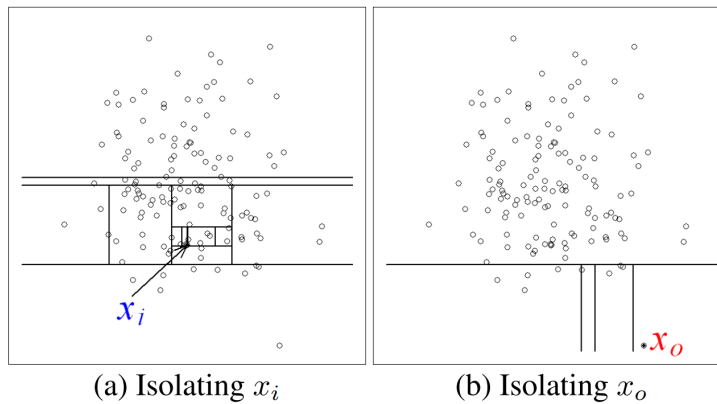


Figure 2.7: A practical example of how the iTree divides the space isolating anomalies faster than normal data points^[24]. Indeed, x_i will need more divisions of the space to be isolated, and, on the other hand, x_o will just need 4 divisions to be isolated.

This isolation characteristic of the tree forms the main support to the method. We call this tree Isolation Tree or iTree. The Isolation Forest builds then an ensemble of iTrees for a given data set, then those instances which have shorter average path lengths on the iTrees are considered anomalies. One of the main aspects is the low number of variables of the method, indeed we can just select the number of trees to build and the sub-sampling size. Algorithms 2.1 and 2.2 show how the IForest is built, Algorithm 2.3 shows how, given an iTree and a data point, the path length used then in the anomalous score formula is computed.

Algorithm 2.1 $iForest(X, t, \theta)$

Input: X - input data, t - number of trees, θ - subsampling size

Output: a set of t iTrees

```

1: Initialize Forest
2: Set height limit  $l = \text{ceiling}(\log_2 \theta)$ 
3: for  $i = 1$  to  $t$  do
4:    $X' \leftarrow \text{sample}(X, \theta)$ 
5:    $\text{Forest} \leftarrow \text{Forest} \cup \text{iTree}(X', 0, l)$ 
6: end for
7: Return Forest

```

Every iTree will have maximum height equal to l and the union of those iTrees will be the final IForest.

Algorithm 2.2 $iTree(X, e, l)$

Input: X - input data, e - current tree height, l - height limit

Output: an iTree

```

1: if  $e \geq l$  or  $|X| \leq 1$  then
2:   Return  $\text{exNode}\{\text{Size} \leftarrow |X|\}$ 
3: else
4:   Let  $Q$  be a list of attributes in  $X$ 
5:   Randomly select an attribute  $q \in Q$ 
6:   Randomly select a split point  $p$  from  $\text{max}$  and  $\text{min}$  values of attribute  $q \in X$ 
7:   Set height limit  $l = \text{ceiling}(\log_2 \theta)$ 
8:    $X_l \leftarrow \text{filter}(X, q < p)$ 
9:    $X_r \leftarrow \text{filter}(X, q \geq p)$ 
10:  Return  $\text{inNode}\{\text{Left} \leftarrow \text{iTree}(X_l, e + 1, l),$ 
11:                $\text{Right} \leftarrow \text{iTree}(X_r, e + 1, l),$ 
12:                $\text{splitAtt} \leftarrow q,$ 
13:                $\text{splitValue} \leftarrow p\}$ 
14: end if

```

For building every iTree, a recursive function is used. It starts from the root and it continues until the height doesn't reach the maximum or the singular data point. Every

division is totally random. The ensemble of the different random iTrees, in this way, will then work on the statistical expectations of the attributes.

Algorithm 2.3 *PathLength*(x, T, e)

Input: x - an instance, T - an *iTree*, e - current path length; to be initialized to zero when first called

Output: path length of x

```

1: if  $T$  is an external node then
2:   Return  $e + 2\ln(T.size - 1) - (2(T.size - 1)/T.size)$ 
3: end if
4:  $a \leftarrow T.splitAtt$ 
5: if  $x_a < T.splitValue$  then
6:   Return PathLength( $x, T.left, e + 1$ )
7: else
8:   Return PathLength( $x, T.right, e + 1$ )
9: end if

```

The *PathLength* function is the standard one used for tree structures, but it is recursive and it uses an adjustment term. The fundamental part is the anomalous score computation. Given an inference data point, the lower the expected distance from the root is, the more the point is anomalous.

In [27–29] it is possible to find some examples of how Isolation Forest can be adopted on use cases where time series are involved. The main approach is based on the use of a window with length w , and on considering all the data points of a given univariate time series inside that window. This, indeed, will be the approach used also in the thesis, with some peculiarities given by the unique use case and by the type of anomalies we search for.

2.4.2. Autoencoders

Autoencoders are unsupervised learning techniques where we adopt neural networks in order to perform representation learning. In particular, as shown in Figure 2.8, the neural network architecture imposes a bottleneck that forces a compressed knowledge representation of the original input. The model assumes that a sort of structure in the data exists, for instance, that there are correlations between some input features. If this is not true, having totally independent input features makes the compression and reconstruction a very difficult task.

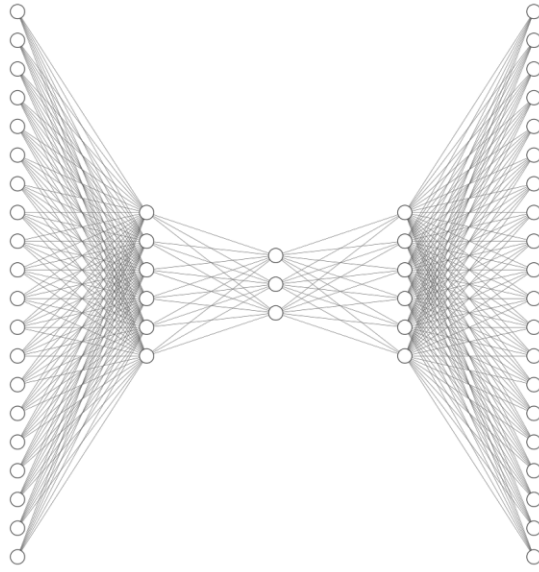


Figure 2.8: Typical architecture of an autoencoder. On the left we have the input layer composed of N neurons, meaning that every instance of our dataset will be composed of values in N dimensions. In the middle, we have the classic bottleneck layers. Here we have fewer neurons because we want to reduce the dimensionality keeping only the remarkable features. In the end, we find the output layer, where the number of neurons is the same as the input.

Given an input x and the consequent output $\hat{x}$, autoencoders are trained by minimizing the reconstruction error, $\mathcal{L}(x, \hat{x})$. In this way, the model will learn how to reconstruct the input considering only the most important features. The bottleneck is the key attribute of this technique because it permits to extract the main features of the data.

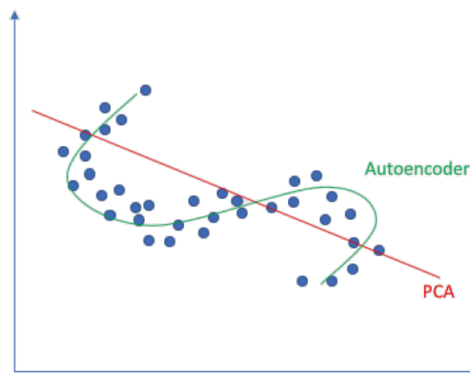


Figure 2.9: Comparison between autoencoder and PCA with respect to the reduction of dimensionality. The Principal Component Analysis searches for a linear hyperplane, the autoencoder searches for a non-linear representation^[30].

Furthermore, as shown in Figure 2.9, neural networks are capable of learning nonlinear relationships. For this reason, autoencoders can be seen as a more powerful generalization of PCA (Principal Component Analysis). Indeed, what the PCA does is to discover a lower-dimensional hyperplane that best describes the original data. autoencoders, on the other hand, are capable of learning nonlinear relations in the data⁵.

After understanding how this kind of architecture work, we must mention the way we can use them for performing Anomaly Detection. We saw that autoencoders learn how to reconstruct the input. We can use this property for designing the anomalous score computation in inference time: if we use almost only normal data in the training phase, the model will be able to reconstruct regular inputs in an accurate way, but we will have a higher reconstruction error in case of anomalous instances [31, 32]. The reconstruction error will be indeed our anomalous score.

Unfortunately, when time-series are involved, the simple versions of autoencoders are not able to perform well and they need some specific preprocessing. It is usually better to use some sort of memory unit-based model able to get in input this kind of data.

LSTM-autoencoder

The Long Short-Term Memory (LSTM) [25] is a recurrent network architecture capable of, in conjunction with an appropriate gradient-based learning algorithm, overcoming the problems of previous similar models related to the fact that the temporal evolution of the back-propagated error exponentially depends on the size of the weights. It can learn to bridge time intervals even in case of noisy input sequences, without loss of short time lag capabilities. The unit used can be visualized in Figure 2.10.

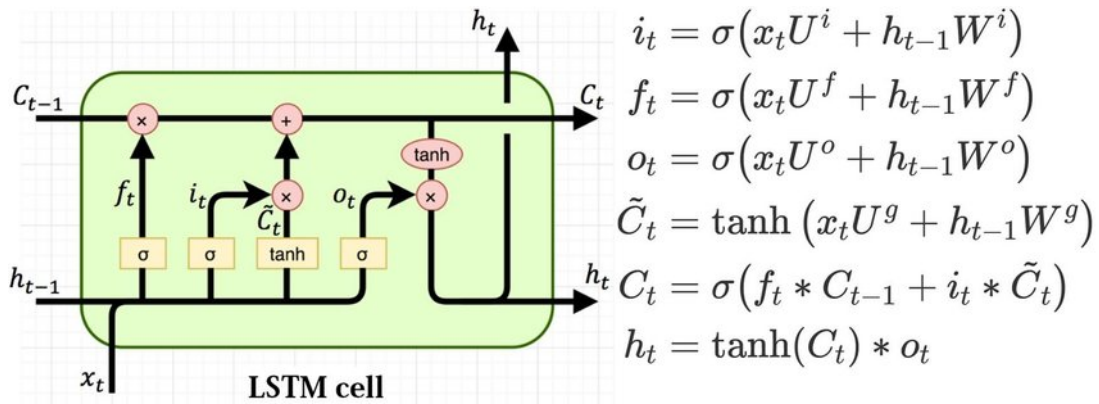


Figure 2.10: Structure of the LSTM cell with the related formulas.

⁵<https://www.jeremyjordan.me/autoencoders/>

Each cell's internal architecture guarantees constant error flow, provided that truncated back prop cuts off error flow trying to leak out. Two gate units learn to open and close access to error flow and the multiplicative input gate offers protection from the noise.

We try to apply this approach to the autoencoder architecture. After all, we still need a sort of network able to reconstruct the input. In Figure 2.11 we can visualize the architecture usually adopted for the LSTM-autoencoder model. Here the autoencoder concept is still valid and the bottleneck is given by the reduced number of LSTM units used in the middle of the structure.

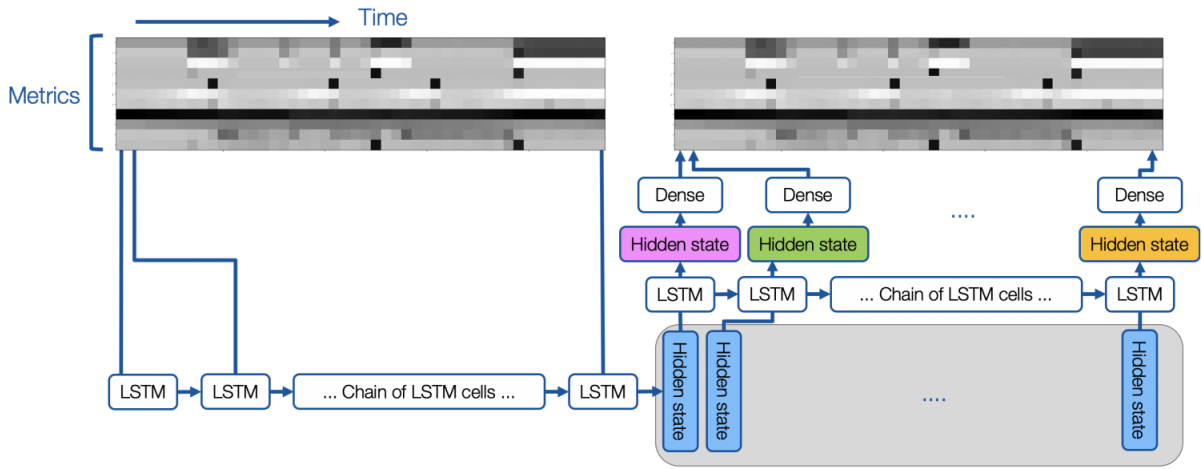
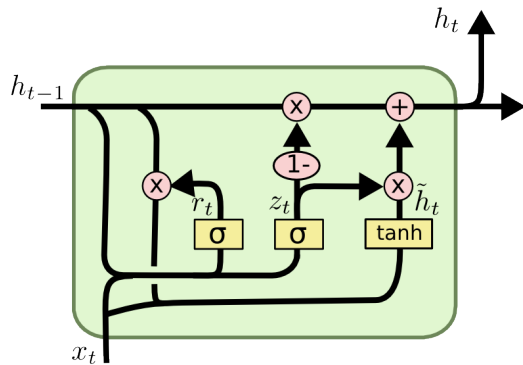


Figure 2.11: Example of a standard LSTM-autoencoder architecture. After the learning process, the input and output sequences should be very similar.

GRU-autoencoder

Trying to improve the LSTM performance, another unit to be used in Recurrent Neural Networks was designed: the Gated Recurrent Unit (GRU) [26]. This unit can be visualized in Figure 2.12.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.12: Structure of the GRU cell with the related formulas.

This unit should be faster than the LSTM one and it should have slightly better performance in general. The architecture in combination with the autoencoder concept is the same as the LSTM-autoencoder, only the used unit changes.

2.5. Related Work and Challenges

Wang et al. [33] studied methods based on Tukey and relative entropy statistics for online detection of anomalies in data centers, but they consider only two metrics: CPU load and Memory. Solaimani et al. [34] proposed a Spark-based AD system for monitoring VMs in a data center, but they use only four metrics from CPU data.

As opposed to Wang et al. [33] and Solaimani et al. [34], we study a larger number and variety of metrics in this work, and the system we present is able, with a little change of the configuration files, to get even more inputs if needed.

Calheiros et al. [35] have studied a dataset recorded from a real data center, considering a larger pool of metrics from disk, CPU, memory, and network. Anyway, they use only a single method, which is the Isolation Forest, and they really focus on the seasonality and on the trends of the metrics.

More recently, Garg et al. [36] proposed a model for data center AD which combines a meta-heuristic and a DL component, but they test it on a simulated dataset where they exclusively used network traffic data. Another line of research is concerned with cyber security anomalies [36–38], where mostly traffic data is monitored.

All these studies show the challenges when we talk about performing AD in data centers. In particular, we can notice that every case is different from the others. In our cloud infrastructure, for instance, the servers' performance metrics don't have any trends or seasonality, and many different services are offered to the users, making the whole data center very heterogeneous. Furthermore, our definition of Anomaly is based on the general behavior of the cloud, not on the behavior of the single machine.

These details, together with the possibility of having many more metrics with respect to the other previous works, make our study necessary for the implementation of such AD system in the CERN cloud infrastructure.

2.6. Summary

In this chapter, we went more in-depth on the major topics related to the project of this thesis. Firstly, we should now have a better overview of what we mean by cloud

computing. It is the first presented subject because the whole project is about this major field. The goal, anyway, is not about cloud computing in general, but it is about building the best possible system for the CERN cloud use case. Because of this, it is also important to know the details of how cloud computing works at CERN. In order to know the basics of how building such a system, we also went into the details about the AD task, looking at which are the objectives and understanding the different ways to perform it. Moreover, being our case based on time series, we finally checked how it is possible to manage those kinds of inputs, especially giving an overview of the main models we will implement and compare in the thesis.

Finally, we presented the studies related to our work. Many papers are cited, but there are not many studies about the specific subject of implementing and applying the previously mentioned AI models on a use case similar to the CERN cloud one using time series metrics in a multi-variate setting.

This will not be the first official study on this specific topic. Another thesis [2], written by the previous Technical Student in the cloud Data Analytics team at CERN, approached the same problem. The study reports a comprehensive comparison between the most known algorithms and models used for performing Anomaly Detection. Moreover, in a previous paper [1], we reported the work done in the first part of this project focusing on the engineering aspect.

Both the previous thesis and the last publication will be at the basis of this thesis approach; for this reason here we are focusing, for instance, on fewer models, selecting the best ones from the last study but trying to improve them.

3 | Proposed Solution

In this chapter, we present our proposed solution. We want to show the details of the approach we used to solve the problem of building the AD system described until now. There exist many possible solutions for dealing with the constraints of the system, but we had to select some specific ones. Here we will explain our decisions providing, to the best of our ability, the motivations that pushed us to make them.

A general overview of our approach can be the following: starting from the previous related thesis work, we selected the algorithms that seemed to work better for this use case, we tried to improve them by studying in detail the kind of inputs we can use, the preprocessing steps and the training phase, we compared them using both experts feedbacks and quantitative evaluations on a labelled dataset, we built a pipeline to run the system daily and we provided a dashboard to the experts to visualize the results produced by the machine and deep learning models.

Given this very small summary, let's see the problem formulation, the pre-processing steps, the actors involved and, finally, the whole architecture of the system.

3.1. Problem Formulation

We model a data centre as a set of computer clusters. A cluster C is composed by N servers having same hardware equipment and software configuration. Each specific server in the the cluster is denoted by S_i with $i = 1, \dots, N$ and generates M time series by recording metrics related to its performance, such as the usage of CPU, memory, disk, network, etc. We denote with

$$s_i^j = \{s_i^j(t), t = T_1, T_2, \dots\} \quad (3.1)$$

the time series of the j -th performance metric for the server S_i acquired at the timestamps t , where t is arbitrarily numbered from 1 to infinite. For instance, the blue lines in Figure 3.1 show, for the same server, the time series related to *disk IO time* and *CPU load*.

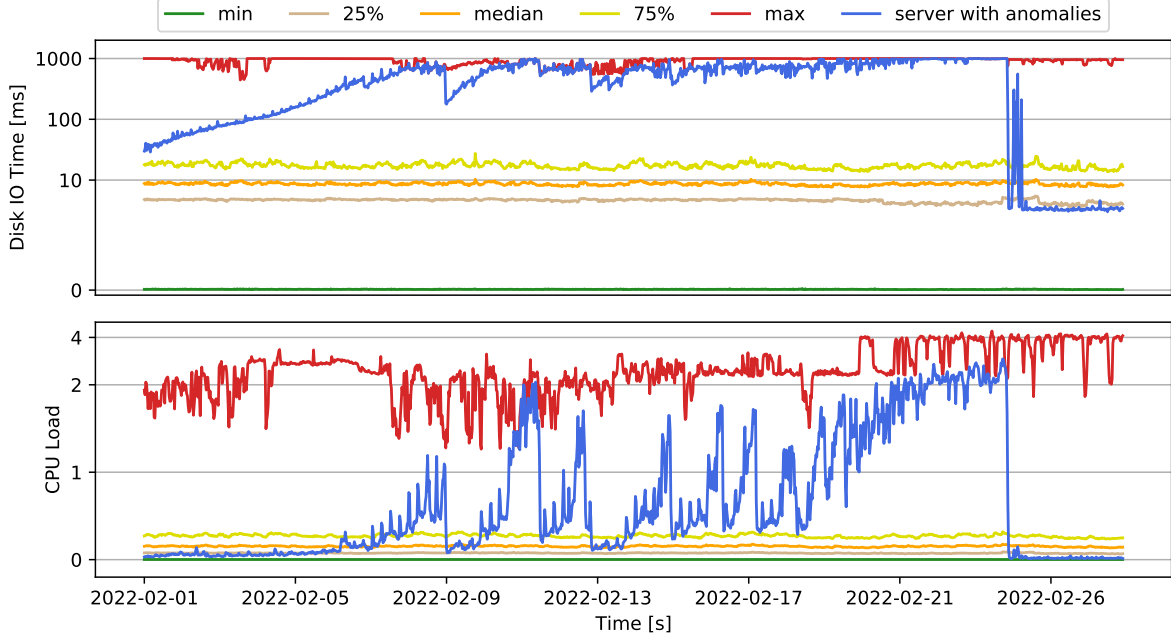


Figure 3.1: Dynamic range of *disk IO Time* (top) and *CPU load* (bottom) for the servers belonging to a same cluster in the CERN cloud infrastructure. The cluster statistics (minimum, 25th and 75th percentile, median and maximum values) are reported, as well as the metric value for a single server (blue curve). Each time series covers one month of monitoring data, with a time resolution of 30 min.

All the time series $\{s_i^j, i = 1, \dots, N, j = 1, \dots, M\}$ might be acquired at different times $T_1, T_2, \dots$, possibly with different sampling frequencies (e.g. *Disk IO Time* is being acquired at a frequency in the order of seconds, while *CPU Load* in the order of minutes). However, we assume that all these are synchronized, namely that they refer to a unique reference clock.

We assume that in normal operating conditions, all the time series are being generated by a process $\mathcal{P}_N$ which is unknown, and which we refer to as normal.¹ When at time τ an anomaly occurs in the data center, the time series are generated by an anomalous process $\mathcal{P}_A$, which is also unknown and different from $\mathcal{P}_N$. The anomalous behavior might affect only a few servers or a few metrics, thus we model our time series as follows:

$$s_i^j(t) \sim \begin{cases} \mathcal{P}_N & \text{if } t < \tau \quad \forall i, j \\ \mathcal{P}_A & \text{if } t \geq \tau \text{ for at least some } i, j \end{cases} \quad (3.2)$$

where we assume that time series s_i^j that are not affected remain generated by $\mathcal{P}_N$ even

¹Here and thorough the thesis, we refer to normal *as being under control*, and we never mean *Gaussian*.

when $t \geq \tau$. We remark that, even though (3.2) describes a change-detection problem, we rather refer to anomaly detection in our work since the process $\mathcal{P}_A$ is typically triggered by a server following an anomalous behavior.

Our goal is to design an algorithm that steadily monitors all the time series $\{s_i^j, i = 1, \dots, N, j = 1, \dots, M\}$ and that promptly reports any anomaly. To this purpose, we assume that a training set TR , containing a portion of the time series from all the N servers of the cluster C , is given to configure our algorithm. TR is not guaranteed to be free from anomalies. However, from empiric evidence we assume that the vast majority of TR is generated by $\mathcal{P}_N$, and anomalies - if any - are very rare. TR is unlabeled, therefore we operate in an unsupervised AD scenario.

3.2. Time Series Pre-processing

One of the most important parts of our solution consists in the pre-processing phase. For addressing our AD problem, we have to process the time series related to the monitoring metrics. Each server produces M time series related to its performance metrics.

The time series under study are subject to three pre-processing transformations, named Downsampling, Normalization and Windowing. This sequence of steps is applied to the training set TR as well as to the upcoming data on which anomalies are predicted.

Downsampling

Given the time series s_i^j related to the j -th acquired performance metric of the server S_i , a new time series $d_i^j = (d_i^j(t_1), d_i^j(t_2), d_i^j(t_3), \dots)$ is derived reducing the original temporal resolution by averaging, namely

$$d_i^j(t_k) = \text{mean}(\{s_i^j(t) \in s_i^j \mid t_k - \Delta < t \leq t_k\}) \quad (3.3)$$

where Δ is the sampling interval. Hence, the set $\{d_i^j, i = 1, \dots, N, j = 1, \dots, M\}$ of all downsampled time series presents the same number of data points aligned in time.

Normalization

Each j -th metric is standardized in order to make all the metrics equally important in our anomaly detection formulation. The standardization uses the mean $\mu_j(C)$ and standard deviation $\sigma_j(C)$ parameters, computed from the data of all servers in cluster C , over the

training set TR :

$$\mu_j(C) = \frac{\sum_{i=1}^N \sum_{t=1}^T d_i^j(t)}{N \cdot T}$$

$$\sigma_j(C) = \sqrt{\frac{\sum_{i=1}^N \sum_{t=1}^T (d_i^j(t) - \mu_j(C))^2}{N \cdot T}}$$

where T is the length of the downsampled time series in TR , N is the number of servers in cluster C and the definition holds $\forall j \in \{1, \dots, M\}$. Subsequently, for each server S_i and each j -th metric, the normalized time series $n_i^j = (n_i^j(t_1), n_i^j(t_2), n_i^j(t_3), \dots)$ is computed as:

$$n_i^j(t_k) = \frac{d_i^j(t_k) - \mu_j(C)}{\sigma_j(C)} \quad (3.4)$$

The parameters $\{(\mu_j(C), \sigma_j(C)), \forall j \in \{1, \dots, M\}\}$ are stored as part of the model, having been computed on the training set TR .

Windowing

The status of a server S_i is analyzed in consecutive, non-overlapping time windows of length $L \in \mathbb{N}$, which define the unit of analysis of our method. The temporal behavior of the server S_i in a window delimited by two timestamps $[t_k, t_{k+L-1}]$ is conveniently represented by the matrix of its downsampled and normalized metrics, referred as the metric window $W_{i,k} \in \mathbb{R}^{M \times L}$:

$$W_{i,k} = \begin{bmatrix} n_i^1(t_k) & n_i^1(t_{k+1}) & \cdots & n_i^1(t_{k+L-1}) \\ n_i^2(t_k) & n_i^2(t_{k+1}) & \cdots & n_i^2(t_{k+L-1}) \\ \vdots & \vdots & \ddots & \vdots \\ n_i^M(t_k) & n_i^M(t_{k+1}) & \cdots & n_i^M(t_{k+L-1}) \end{bmatrix} \quad (3.5)$$

where $n_i^j(t_k)$ is defined in (3.4). The evolution of the server status is therefore captured by the series $\{\dots, W_{i,k-L}, W_{i,k}, W_{i,k+L}, \dots\}$.

This matrix representation is important also because it makes it possible to use algorithms that can model the temporal dimension, for instance, it is the case for the autoencoders models. It has also a different visual representation used to inspect data with a heatmap plot (Figure 3.2).

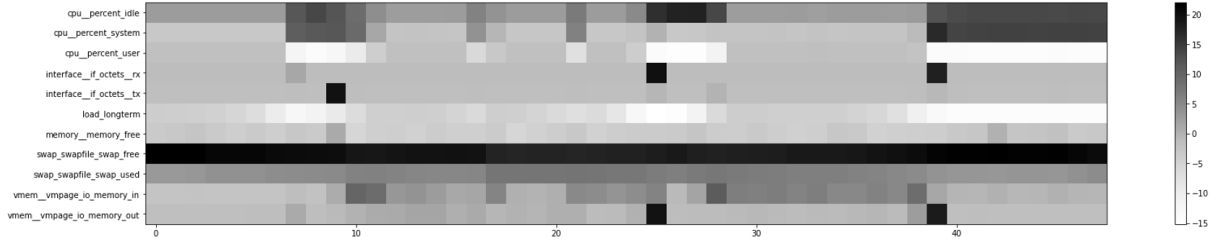


Figure 3.2: Heatmap representation, in grey tones, of a metric Window for a single hypervisor. On the y-axis, the labels identify the metrics, on the x-axis we have the time^[2].

3.3. Anomaly Detection Approach

We approach the detection of servers having an anomalous behavior in (3.2) by learning an anomaly score function $\mathcal{A}$ that takes as input a window $W_{i,k}$ and returns positive scores close to 0 for $W_{i,k}$ generated by $\mathcal{P}_N$, and higher scores for $W_{i,k}$ containing values generated by $\mathcal{P}_A$. This is a rather usual approach in the literature, where the anomaly score function is a mapping of this type:

$$\mathcal{A}(\cdot): \mathbb{R}^{M \times L} \rightarrow \mathbb{R}^+ \quad (3.6)$$

that analyzes the collective behavior of all the N servers in the cluster C and is trained on the whole training set TR . We detect anomalies by simply comparing $\mathcal{A}(W_{i,k})$ against a threshold Γ as follows:

$$AD(W_{i,k}, \Gamma) = \begin{cases} 1 & \text{if } \mathcal{A}(W_{i,k}) > \Gamma \\ 0 & \text{otherwise} \end{cases} \quad (3.7)$$

The threshold Γ is typically set to control the false positive rate, i.e. how often normal windows are being considered as anomalous. In particular, Γ can be selected as a quantile of the anomaly score from a subset of windows from TR that have not been used for learning $\mathcal{A}$.

We have considered three unsupervised methods to model three different anomaly score functions $\mathcal{A}_\lambda$ ($\lambda = 1, \dots, 3$), namely: Isolation Forest (IFOR), Long-Short Term Memory autoencoder (LSTM-AE) and Gated Recurrent Units autoencoder (GRU-AE).

Isolation Forest

IFOR [39] is an anomaly detection algorithm that identifies anomalous samples based on how much these can be isolated from densely populated region containing normal data. IFOR, is not naturally designed to model the temporal behavior of a multivariate time series, however, we concatenate the rows of each window $W_{i,k} \in \mathbb{R}^{M \times L}$ into a vector $\vec{w}_{i,k} \in \mathbb{R}^V$ with $V = M \cdot L$, and use this as an input to IFOR. Except for this aspect, we use the off-the-shelf implementation of IFOR. During training, a predefined number of data structures (iTree) is constructed by randomly partitioning TR . During inference, a test window $\vec{w}_{i,k}$ is fed to each iTrees, keeping track of the leave where this falls, which denotes how easily $\vec{w}_{i,k}$ can be isolated from normal data. The average depth of these leaves is computed and used to assess the anomaly score $\mathcal{A}(\vec{w}_{i,k})$ as in [39] equation (2).

LSTM and GRU autoencoders

The other two anomaly score functions are obtained by a different approach, which is based on training a neural model to describe the multivariate time series in TR . Each input window $W_{i,k}$ to be tested is reconstructed using the trained model, and we take as the anomaly score the reconstruction error. The rationale behind this approach is that windows generated from $\mathcal{P}_A$ would be poorly reconstructed from a model trained to reconstruct windows from $\mathcal{P}_N$, thus would result in higher reconstruction errors than normal windows. In contrast with threshold-based methods and IFOR, these solution learn the temporal dynamic of the time series, thus have the potential to detect anomalous patterns that do not result in outlying values.

In our experiments, we train two neural autoencoders [40]: one based on Long-Short Term Memory (LSTM) [25] and another on Gated Recurrent Units (GRU) [26], which are successful deep learning models to describe multivariate time series. More specifically, each window $W_{i,k}$ undergoes an additional normalization to shrink the min and max values in a fixed range $[0, 1]$, resulting in $\widetilde{W}_{i,k}$. Then, the window $\widetilde{W}_{i,k}$ is fed, one column at a time, to the trained autoencoder $AE(\cdot)$ which at the end predicts the reconstructed window $AE(\widetilde{W}_{i,k})$. The anomaly score is the mean squared error of the reconstructed windows, which can also be written as:

$$\mathcal{A}(W_{i,k}) = \|\widetilde{W}_{i,k} - AE(\widetilde{W}_{i,k})\|_F \quad (3.8)$$

where $\|X\|_F$ denotes the Frobenius norm of a matrix, namely the ℓ_2 norm of the vector $\vec{x}$ of the unfolded rows.

We tested different autoencoder architectures, either containing one or three LSTM/GRU cells in the encoding part, and then a symmetric shape for the decoder. At the bottleneck, each sample of the M -dimensional time series is condensed to a scalar. At the end of the architecture a dense layer is used to adjust output sizes. Note that a dropout is applied as a form of regularization after each LSTM/GRU layer.

3.4. Ensemble

Since different machine learning models are able to learn different patterns characterizing normal windows, we decide to aggregate the three anomaly score functions $\mathcal{A}_\lambda$ in an ensemble. In order to equally weight the three scores $\mathcal{A}_\lambda$ and overcome issues arising from them having different ranges, we define the detection output of each ensemble (ENS_e) by aggregating the individual outcomes through voting strategies, i.e. by counting how many individuals trigger a detection, thus satisfy: $AD_\lambda(W_{i,k}, \Gamma_\lambda) = 1$ for a given metric window $W_{i,k}$:

$$ENS_e(W_{i,k}) = \sum_{\lambda} AD_\lambda(W_{i,k}, \Gamma_\lambda) \geq e, e = 1, \dots, 3 \quad (3.9)$$

These ensembles correspond to the implementation of the voting strategies ‘OR’, ‘MAJ’ and ‘AND’ for $e = 1, \dots, 3$ respectively. In order to tune the false positive rate of the ensemble, it is convenient to adjust simultaneously all the thresholds Γ_λ , considering only the values which lead to the same false positive rate for the individual models. It is worth remarking that, with $e > 1$ it is often possible to relax the individual thresholds Γ_λ , yet guaranteeing a stricter control over false alarms for the ensembles.

3.5. Actors

Before moving on, we want to recap who are the actors of our system. They will be needed during the description of the architecture.

We can identify 3 major actors involved in the AD system: cloud users, cloud managers and AD system managers. In Figure 3.3 a visual scheme of the actors and their related services is shown. A deeper explanation of the 3 groups follows.

- **Cloud users:** these are those employees at CERN who are, in one way or another, tied at the cloud resources. There are many different kinds of users being part of this class:
 - People not related to the IT who anyway use cloud resources to work. For

instance, administrative employees and students use services like *EOS*², which runs in the CERN cloud. Or, for example, the physicists working in the organization are not directly related to the cloud, but they use cloud services for their experiments (e.g. *SWAN*, *GitLab*, ...).

- People who don't use resources but just access some CERN web services. For instance, the CERN websites are hosted in the cloud and if some anomalies make them unstable, those kinds of users are not able to access the websites.
- People who work from home. For instance, for doing that, they usually need to access the CERN *lxplus*³ service, which runs in the cloud infrastructure too.
- People who work in the IT department. Those employees are deeply involved in the IT infrastructure of the organization and they are surely affected if some services stop working properly.

Cloud users consist of the group of people in general affected by hypothetical anomalies and they are the ones who, in case of misbehavior, realize that something is wrong and create a ticket for the cloud managers.

- **Cloud managers:** these are those employees who literally work on the cloud infrastructure. They usually are part of the IT department in the Compute and Monitoring group, and they have special access to the cloud for management and implementation purposes. The cloud managers are the ones who read the tickets sent by the cloud users, trying to figure out the problem and how to solve it.

Cloud managers are the real users of our system: cloud users will not be able to access it, they don't need it and they shouldn't be able in any case to see so deep information about the infrastructure. On the contrary, the managers will use the system on a daily basis to discover new possible misbehaviors, trying to prevent any future impact on the cloud users.

- **AD system managers:** these are the employees working on the system we are talking about. In particular, we are the managers of the system and it is our duty to make it work in a proper way. This includes avoiding as much as we can the detection of false positives and presenting the anomalous results using a dashboard easy to consult.

²<https://eos-web.web.cern.ch/eos-web/>

³<https://information-technology.web.cern.ch/book/lxplus-service/lxplus-guide>

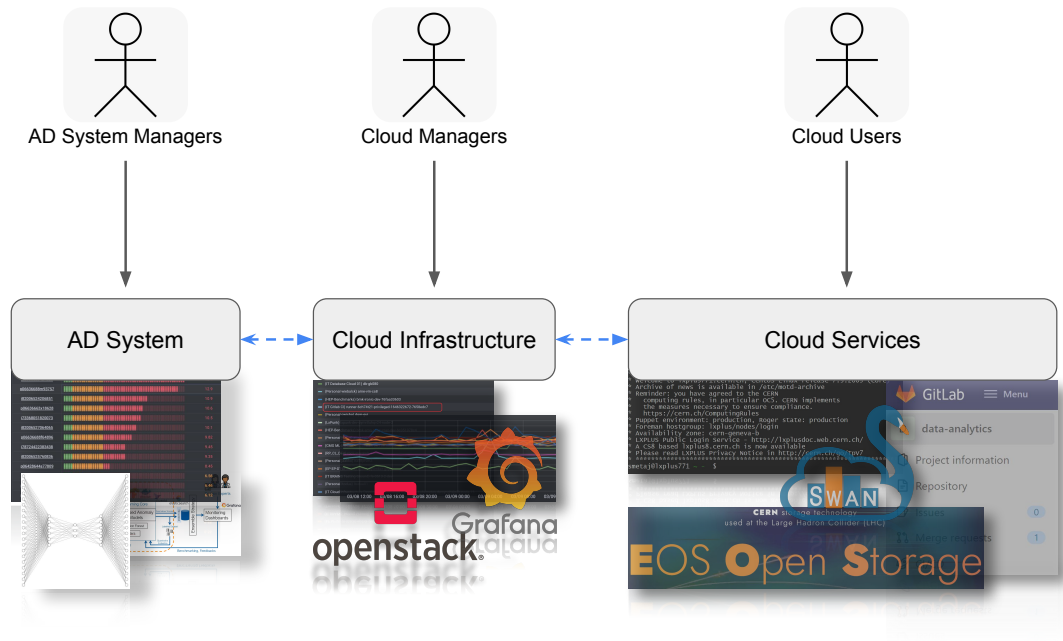


Figure 3.3: The main actors of the CERN cloud. Services are offered by the infrastructure, the AD system takes the information from the infrastructure to train the models and for computing the anomalous scores.

3.6. Architecture

In this section we will talk about the architecture of our system, mentioning also the main technologies and tools adopted. We want to see only the main ones here, leaving the detailed list for Chapter 4.

We will try to summarize all the information gathered until now to explain, in its totality, how the pipeline and its phases work, and to see more details about the pipeline orchestrator and the jobs schedule.

3.6.1. Pipeline

In Figure 3.4 it is possible to visualize the pipeline we are talking about. Every major phase of the pipeline is represented by a box, and the arrows represent the flow of data, results or feedback. Over each phase, we can find the technologies and tools behind that specific step. As we say in the caption of the figure, it is also important to notice the 2 loops that we have, because they help us in the process of improving the results. However, they are different and they have different origins:

- The first one, larger in the image, comes from the feedback of the experts (the cloud

managers). If they notice, during the daily operations, some problems with the results, or the presence of false positives, which we hardly want to avoid, we get notified. In that case, we start to study the models using the new information in order to improve them.

- The second one, smaller, comes from the experiments we do with the evaluation dataset. The process for computing the anomaly score is the same as before, but instead of sending them to the CERN infrastructure, we use them just for computing the metrics that we need for the evaluation.

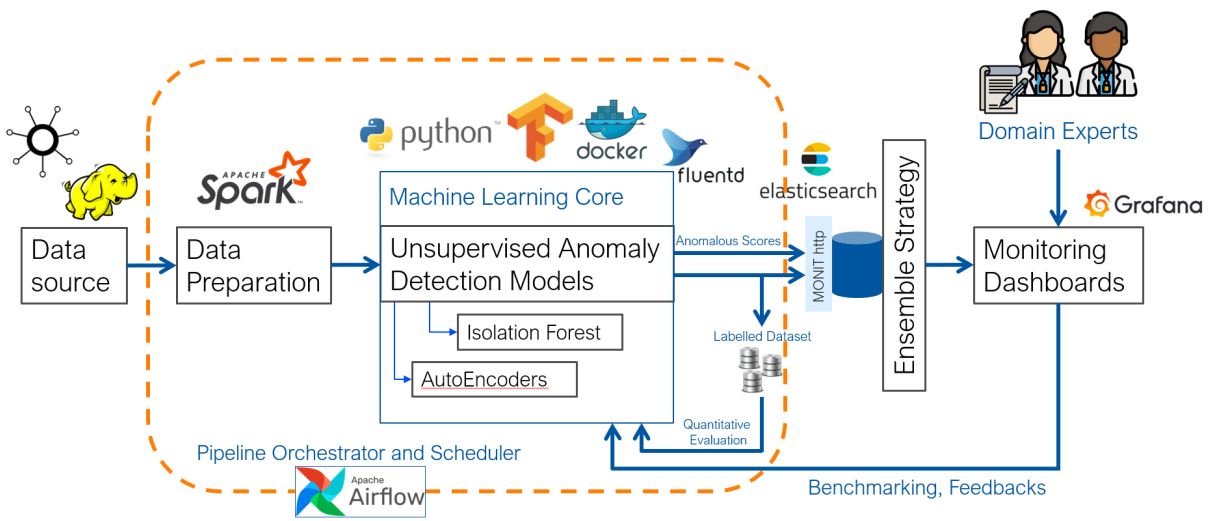


Figure 3.4: The complete conceptual pipeline of our AD system. Notice that we start from the data source, moving to the experts’ feedback, which create a feedback loop aimed to improved our ML core for having better results.

Notice also that the major part of the pipeline is under *Airflow*’s control. That’s indeed its job and, besides the storage of initial data and the management of the final results, everything is “orchestrated” by it.

Let’s focus now on each of the phases to describe better what happens.

1. **Data source:** here we don’t have so much control. Everything is already managed and done by the central monitoring infrastructure with the help of *Collectd*⁴ and *HDFS*⁵. In particular, *Collectd* aggregate all the data it is able to get from the hypervisors, and it saves them in the Hadoop File system used at CERN.
2. **Data preparation:** this is one of the two most important phases of the whole pipeline, where we actually have all the supervision. The AD models we are consid-

⁴<https://collectd.org/>

⁵https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

ering are unsupervised, because of the lack of labels and because of our definition of anomaly. This means that the data pre-processing step is fundamental and directly related to both the performance of the models and the goodness of the results.

For the data preparation, we use *Apache Spark*⁶, which allows us to retrieve the data in a distributed way from *HDFS*, and to process the data thanks to its many features and functions.

3. **ML core:** after the processing of the data, we have now a clean and optimized dataset to feed our Machine and Deep Learning algorithms. In particular, the first time the pipeline is launched, we train each of the models included in the configuration file, and we save them in the cloud storage. Once the models are saved, we use them to run the inference step, which finally produces all the results.

Notice that we don't make difference between models belonging to the traditional methods and the ones belonging to the DL family: thanks to a *PyOD*⁷ wrapper class, we are able to call the same methods for both models of the *PYOD* official library and personalized models implemented in the wrapper. This wrapper class is also the one that decides the structure of the JSON to print out.

4. **Results storage:** this is the phase right after the computation of the anomaly results. The wrapper class includes in the JSON the anomalous scores and information about the specific hypervisor, the used algorithm and the input metrics. Created the structure, and filled it, the JSON is printed by the class. The *Fluentd* service, which is continuously checking the outputs of the system, recognizes the JSON structure as the one it has to send, and it injects the file into the *ElasticSearch* database used by the monitoring team.

Now, from the monitoring services used at CERN, we are able to get the results previously injected, including the possibility to apply some ensemble strategies in the case of multiple models agreeing about the anomalousness of the same time window.

5. **Monitoring dashboards:** finally, we have our anomaly results in the *ElasticSearch* infrastructure. In this phase, the experts and ourselves check those results thanks to the AD system monitoring dashboards, which we will describe more in detail in the next section. In general, dashboards, in the CERN monitoring team, are designed and visualized through *Grafana*⁸. This tool permits to building a huge number of

⁶<https://spark.apache.org/>

⁷<https://pyod.readthedocs.io/en/latest/>

⁸<https://grafana.com/>

different plots starting from data stored in different databases. A dashboard is just a specific set of specific plots, usually referring to the same class of information, as described in Definition 1.1.6.

It must be said that, as easy to imagine, our system is dependent on all the services and infrastructures we mentioned, with different effects with respect to different services. For instance, if the *HDFS* service is broken, we can't even get the raw initial data, meaning that the system can not produce results at all. On the opposite, if *Grafana* is broken for instance, the system still gets and process the data producing the results, but we are not able to see the usual dashboards (in that case, for example, we can use other services like *Kibana*⁹, another dashboard framework strictly connected to *ElasticSearch*).

3.6.2. Orchestration and Scheduling

We want to describe here *Apache Airflow*, the tool we use for orchestrating the pipeline and for scheduling the jobs. It is one of the most important parts of the system, and it allows us to have control over the time in which we run our computations and over the parallelization of the tasks. Moreover, in order to have a transparent view of our implementation in *Airflow*, we will show some figures related to it.

Airflow DAG

Airflow is based on the definition of Direct Acyclic Graphs (DAGs). Those DAGs define the flow of the computation, giving an order to the tasks to be run. Our DAG is shown in Figure 3.5.

This pipeline follows the conceptual one, but it leaves outside all the steps related to the managing of results and related to the visualization of the dashboards. Indeed, in Figure 3.4, we can see that *Airflow* doesn't contain those two phases.

Every colored rectangle is a single task. Single tasks can be grouped in larger blue rectangles. Despite the “dag_exit_status”, used only to see if the DAG failed or not, it is possible to identify 4 different task groups:

1. **TRN**, for the preparation of the training data,
2. **TRN_**, for the training of the model,
3. **INF**, for the preparation of the inference data,
4. **INF_**, for the inference and the publishing of the scores.

⁹<https://www.elastic.co/kibana/>

In particular, the two task groups related to the data preparation have a longer series of tasks and a specific logic based on the success or the failure of the tasks themselves. Every specific task runs in a containerized *Docker* environment taking in input the same configuration files.

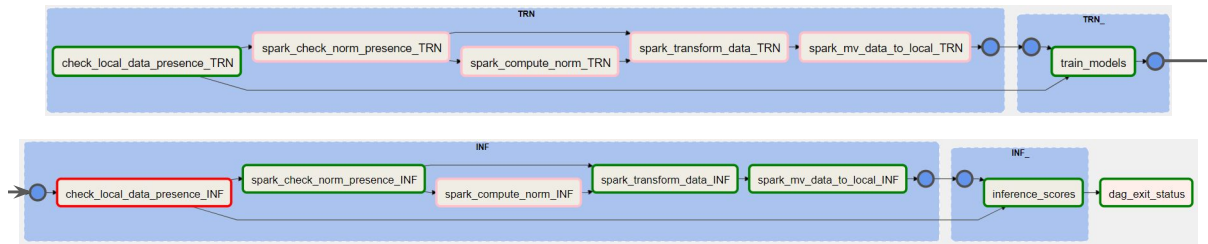


Figure 3.5: The *Airflow* DAG we run.

Let's describe better every task in order to understand better the logic:

- **check_local_data_presence**: this task checks if we already have the data we are interested in. For instance, the training data is usually missing just at the first run of the pipeline, while for every other run of the same pipeline, there is no need to re-prepare the dataset. Another example is about the possibility of multiple runs of the pipeline related to the same period. Also in this case, we already have the data related to both the training and the inference parts, and we want to avoid the run of the whole process again.

If this task succeed, we go directly to the next task group, training the models. If it fails, we proceed with the next task of the same task group.

- **spark_check_norm_presence**: if the previous task fails, it means that we need to prepare the data. The first step is to compute the normalization coefficients and save them in *EOS*. In particular, for every plugin we use, we compute the mean and the standard deviation. Before doing that anyway, we want to check also in this case the presence of those coefficients: maybe we already did the computation. Also here, if the task succeeds we skip to the transformation of the data, while if it fails, we proceed with the next task.
- **spark_compute_norm**: if we don't have already the normalization coefficients, we compute them and store in our *EOS* shared folder. From there, they will be

available to be used by any other task of the pipeline.

- **spark_transform_data**: once we are sure that the normalization coefficients are computed and stored, we finally transform our data. The job of this task is to get the data from *HDFS* and transform them in order to apply the normalization step, and to have the same format we saw in section 3.2. The specific steps inside this task can be found in the Appendix ???. The whole computation of this task happens in *HDFS* using *Spark*, and nothing is saved locally, we store the final dataset again in *HDFS* for isolation purposes.
- **spark_move_data_to_local**: once the transformation is done, we just need to clean the dataset (e.g. removing null values) and save it from *HDFS* to *EOS*, where it will be available to all the other pieces of our system.

This list explains the behavior of the data preparation phase. Notice that a task is green if it was successful, red if it failed, and pink if it was skipped.

We run this pipeline for both training and inference. In particular, the training dataset *TR* contains 1 month of data, selected looking at the statistics of the cloud from a very high level, and the inference dataset will contain 1 day of data. That's because, how we will see in the next paragraphs, we run the pipeline every day.

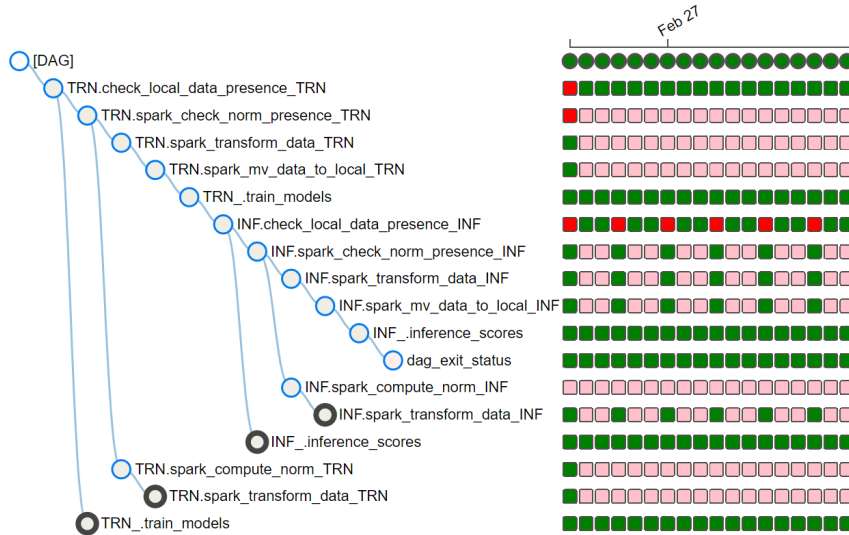


Figure 3.6: The schedule of our AD pipeline. The *Airflow* DAG started, after some modifications, the 25th of February.

Let's also briefly describe the details of the last 2 tasks we didn't mention:

- **train_models**: once we have our data in *EOS* we can train our models. Inside this task, we have the same logic as before. Indeed, we check if we already trained

the models and saved them. If we didn't, we do that. If we did, we just skip to the next phase.

- **inference_scores**: once we have our trained models and our inference data, we are ready to compute the anomaly scores and publish them to the cloud *ElasticSearch* service. Notice that we don't check anything before doing that because, even if we publish the same scores related to the same time windows, the *ElasticSearch* documents are overwritten.

Daily Execution

Once we have built our desired DAG, we have to decide its schedule¹⁰. It is possible to see an example of scheduling in Figure 3.6.

As we said, we are getting the data from *HDFS*. Here there is some delay and we don't have unfortunately real-time data because of the number of monitored machines and because of technical constraints. Looking at the average time we need to wait to have the data ready, we found out that in 95% of the cases, at 8 AM of every day the data related to the previous day is ready to use in the Hadoop File system.

For this reason, we decided to run the pipeline 3 times per day: at 8 AM, 4 PM and midnight. With this schedule, almost every time, we have the anomalous scores related to the previous day published before the working hours start. If this doesn't happen, it means that there is a delay in the *HDFS* saving process and that we have to delay our process too. The midnight run is used 0.001% of the time and it could be avoided, but we couldn't for *Airflow* constraints.

Following this logic it is easy to understand the image in Figure 3.6:

- On the left part we have all the steps to be followed from the check of training data to the dag exit.
- In the first run, the presence check of both training and inference data fail. On the opposite, in all the other runs we already have the training data.
- Focusing on the runs on a specific day, the same logic is applied to the inference data. At 8 AM we never have those data and we have to run all the data preparation tasks, while in the other 2 runs of the day we can skip all those steps.

The difference between the first and the other runs can be also seen in Figure 3.7. Airflow allows us to see, for every specific run of a certain DAG, the related Gantt chart. In

¹⁰<https://airflow.apache.org/docs/apache-airflow/stable/dag-run.html>

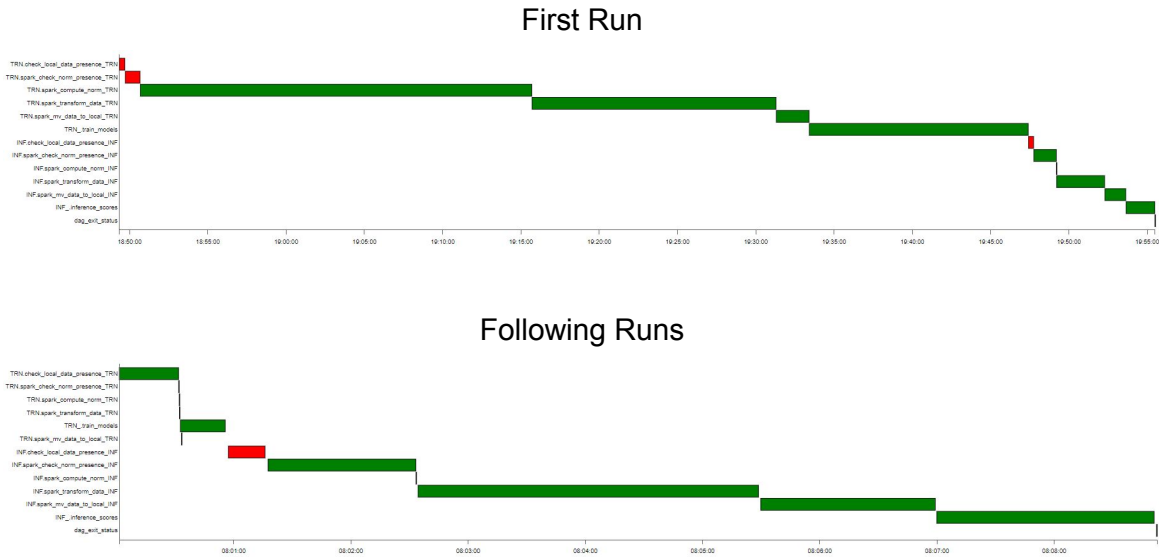
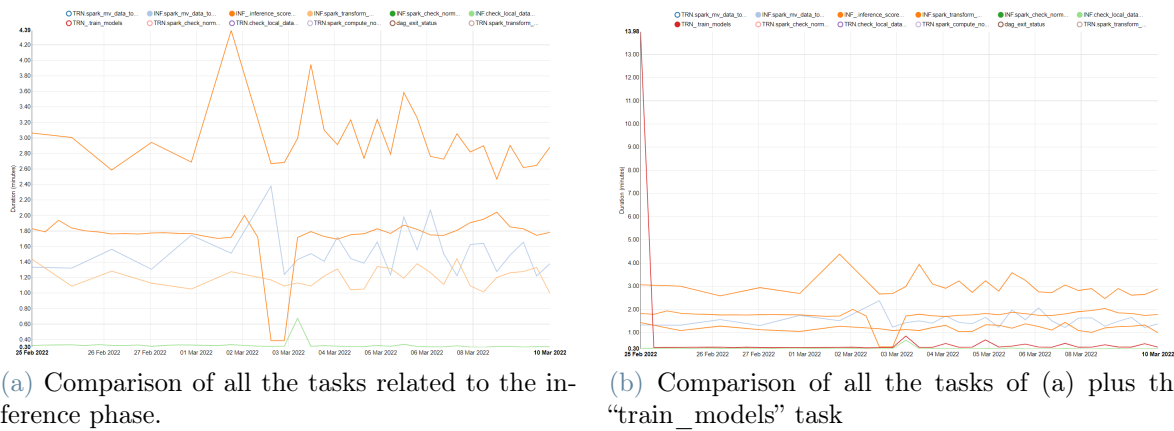


Figure 3.7: Gantt chart related to the first run and the following ones.

particular, in this way, we can visualize the duration of the tasks, if they failed or not, and the time of start and time of the end.

The same information can be visualized in Figure 3.8. In particular, Figure 3.8a shows the time spent for each task of the inference part, Figure 3.8b shows the same, but with the addition of the “train_models” task. The comparison highlights the difference between the first and the other runs, showing that in the first one the “train_models” task took 13 minutes to finish, while the next time it took some seconds.



(a) Comparison of all the tasks related to the inference phase.

(b) Comparison of all the tasks of (a) plus the “train_models” task

Figure 3.8: Duration of the tasks of the AD pipeline. In the x-axis we have the chronological runs, in the y-axis the time in minutes and every line is a different specific task.

3.7. Monitoring Dashboards

This final part of the chapter is related to the monitoring dashboards we developed for allowing the cloud managers to check the results of the system in the easiest and fastest possible way.

The main idea is to highlight the top anomalous hypervisors allowing also to visualize the trend of the anomalies. For doing this, two different dashboards are created:

1. The first dashboard, in Figure 3.9, allows the managers to visualize those hypervisors that are considered anomalous by all the three models we run in production. In particular, for every model, for every time window of 4 hours, we publish the top 20 anomalous hypervisors, and in the dashboard, those hypervisors are listed from the most to the least anomalous.

Thanks to this logic, we are able to allow a fast inspection of the most important problems reported from the models, and to implement an ensemble mechanism using majority voting between the algorithms. Notice that the ensemble is not algorithmic, and we will not perform proper experiments with the final ensemble system.

Host ▾	Distinct Algos ▾	25% rank_position ▾	50% rank_position ▾	75% rank_position ▾	max rank_position ▾	Count ↕ ▾
78724427758663	3	4	5	6	12	54
78724422383438	3	1	2	3	4	54
p06636663d11312	3	7.25	11	12.8	17	47
p06636663c86794	3	7	10	13	17	45
p06636663c78481	3	8	9	11	19	41
p06636688u48653	3	8	9	11	17	38
78724426218616	3	3	6	8	17	37

Figure 3.9: The Grafana monitoring dashboard is related to the ensemble approach. We don't focus on the score values but on the rank of the anomalies.

2. The second dashboard, in Figure 3.10, allows the managers to visualize more details about the results of a specific model. The ensemble approach is a perfect solution for having a fast response, but in order to inspect the anomalous scores published by the models, we need this further dashboard. Here we focus more on the absolute values of the scores.

For a specific hypervisor, we can see how much anomalous it is with respect to the others, and, furthermore, we can inspect the score values related to the last days. This permits to understand the behavior of the server itself and to visualize the trends of the anomalous scores.

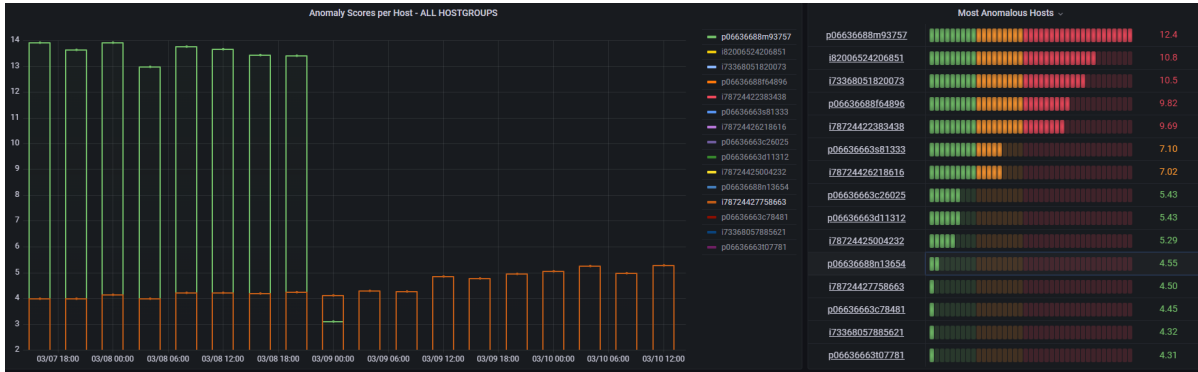


Figure 3.10: The Grafana monitoring dashboard related to the results of a single model. Here we focus on the anomalous scores allowing us to visualize the trends of the anomalies.

These dashboards are not only used by the cloud managers in their daily operations, but also by us, the AD system managers. Indeed, we can inspect the results to check how many false positives we have, to understand how we can improve the models, and count how many more anomalies we are finding with respect to the static threshold-based alerting mechanisms.

4 | Implementation Details

Let's now go deeper into the implementation, focusing on details not mentioned before. In Section 1.2, we already saw the typical steps we have when an anomaly is present in the CERN cloud infrastructure. In that case, the anomaly is not really detected by the system, but its consequences have an impact on the user experience, starting a long procedure in order to notify the managers who will then try to solve the issue.

Here we will show some more detailed scenarios of possible anomalies, we will present our design choices, explaining in depth all the tools we use in the system. In order to understand them, we will also list the goals and requirements of the system.

4.1. Anomalous Scenarios

We present here 2 examples of scenarios and situations where our system can bring an improvement.

4.1.1. Scenario 1: GitLab CI VM Slowdown

Let's suppose to have a given hypervisor of the cloud. Thanks to the virtualization mechanism, many VMs (Definition 1.1.3) are running in the machine, sharing the provided resources (Figure 4.1). The VMs are different and they are used for different purposes. Some of them are personal VMs of many CERN employees, some of them are running benchmarks experiments, others are running CERN services like *Gitlab*¹.

In particular, the *Gitlab* machine is running the CI/CD testing jobs related to a specific project of CERN. The user owning this VM finds that the CI/CD jobs take 20 times more time to finish with respect to before. Something is clearly wrong, but he/she actually didn't modify anything that motivates this change in the performances. After working half a day to understand the problem, he tries to contact the cloud managers opening a ticket, explaining the problem and giving the information about his VM.

¹<https://gitlab.cern.ch/>

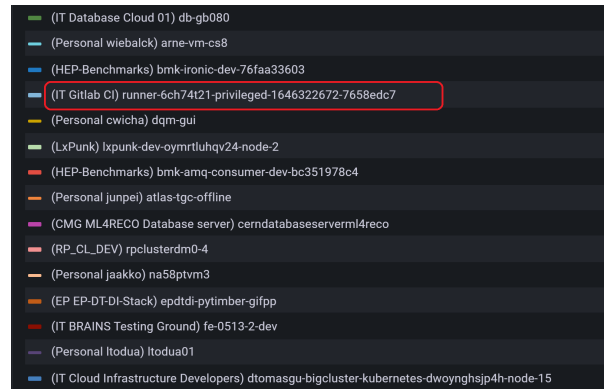


Figure 4.1: The list of VMs in the target hypervisor of the first scenario. We have many services and VMs all together sharing the resources of the same machine. The specific IT *Gitlab* CI VM is at the center of this scenario.

The cloud manager who is working on the user tickets that week, starts the process to understand and solve the problem, looking in particular at the hypervisor monitoring dashboard, shown in Figure 4.2.

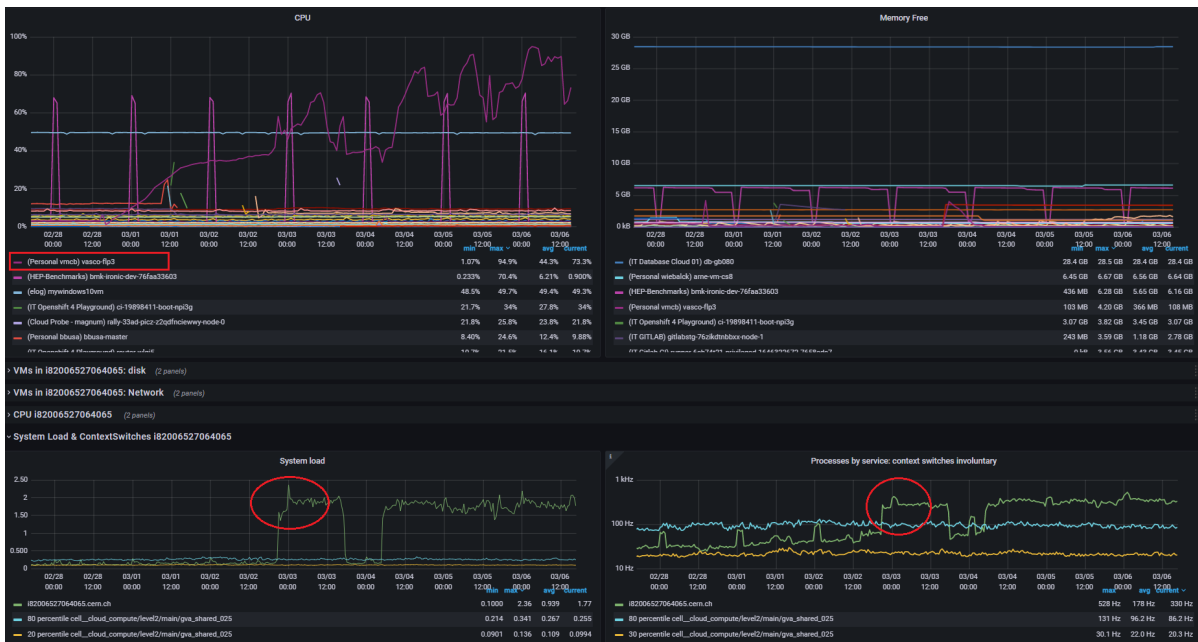


Figure 4.2: The monitoring dashboard of the same hypervisor. We can see that the metrics related to the load and the context switches had a huge change days before. Moreover, a specific personal VM is consuming much more CPU than usual.

The manager, checking the dashboard, finds out that the *Gitlab* VM related to the ticket shares resources with many other VMs, and between them, there is a personal VM consum-

ing many more resources related to CPU, disk and memory than usual, causing problems to all the other services hosted in the same hypervisor. The manager now has to contact the user of the personal VM asking to check its behavior, to wait for the answer, and, when the behavior of the personal VM is back to normal, to send a message to the starting affected user giving information about the machine going back to normality.

4.1.2. Scenario 2: Increasing Disk IO Time

In this other scenario, we are not trying to understand the process of how anomalies are spotted without a proper data-driven system. Indeed, at this point, it should be clear how that works and why we want to avoid it (Section 1.2).

Here we want to do one more step in the direction of preventing possible future problems, talking about some problems our system could prevent. Let's analyze, for instance, the dashboard in Figure 4.3.



Figure 4.3: The monitoring dashboard of another hypervisor. Here both load and disk IO time increased their values. The drop of the plugins is due to a cloud manager intervention. After some users' tickets, and after the previously described process, the manager was able to solve the issues of the machine.

It is easy to see the increase in the absolute value of the metrics, a typical signal of some misbehavior. The main point, anyway, is the time that passed from the start of the misbehavior to the resolution of the issue. Around 20 days were needed for the whole process to finish. This is, indeed, one of the main reasons behind the building of our system. We want to prevent those situations where for many days we have not reported issues in personal VMs and CERN services. On the other hand, it is also crucial to understand if this was really possible, looking at the metrics' values at the starting phase

of the issue (Figure 4.4).

We can see that, weeks before the complaints from the users, the server’s metrics were already increasing their values. In particular, the *disk IO time* metric was facing a really huge increase, going very close to his saturation value, equal to 1.

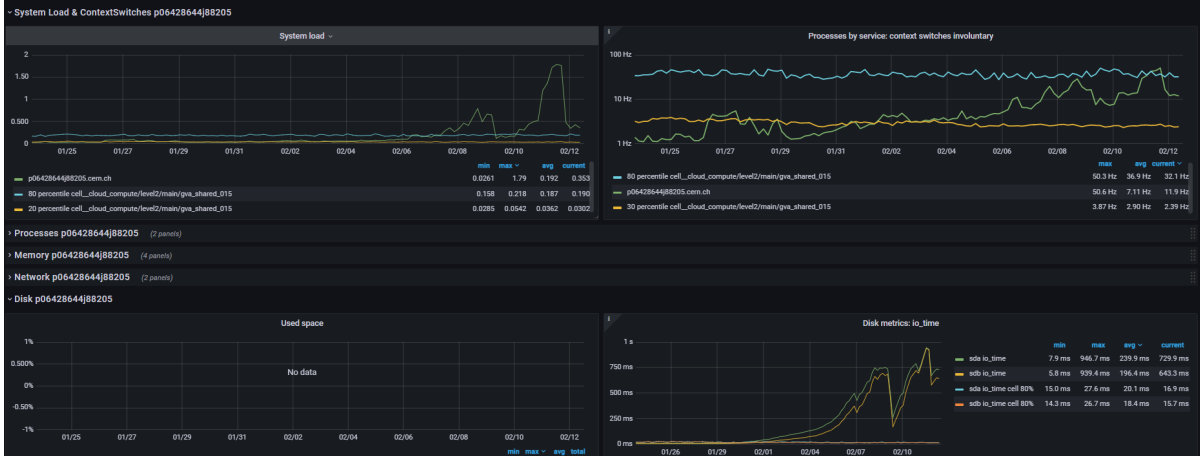


Figure 4.4: Here we have the starting phase of the anomaly. Context switches keep increasing; the CPU load increases too but does not reach super high values; the disk IO time, on the opposite, increases a lot.

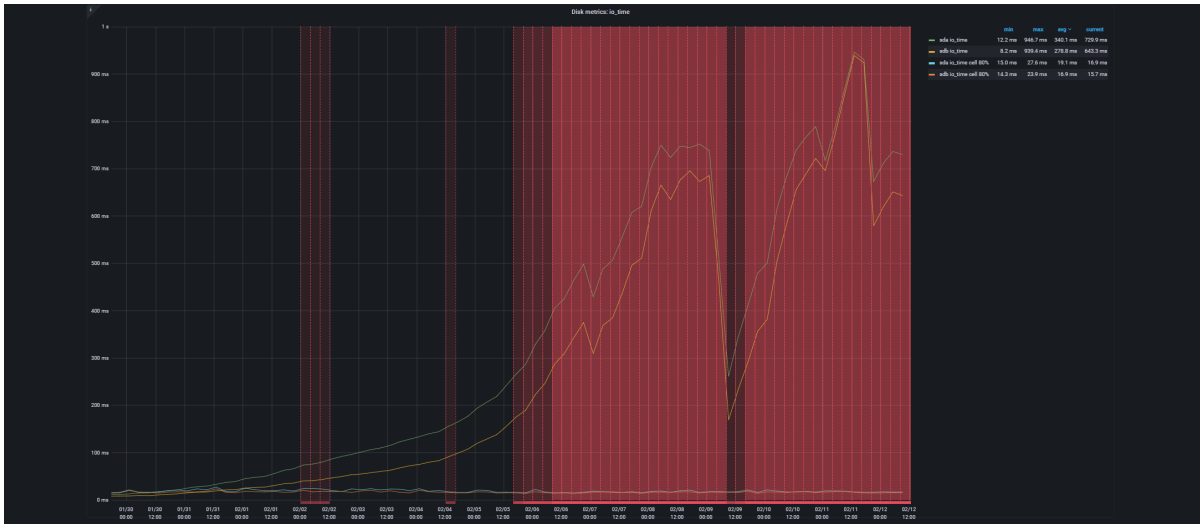


Figure 4.5: Our system detected the misbehavior weeks before the users. Notice that the intensity of red annotations increases with the increase of the number of models detecting the time window as anomalous (in this case, the first model detecting the anomaly was the Isolation Forest; after 1 day of data also the autoencoders started to detect it).

Trying to run our system on this case, as shown in Figure 4.5, it was able to detect this

kind of anomaly. Moreover, using different models with an ensemble approach, we are able to prevent those anomalies with increased robustness.

Here we can also see all the improvements with respect to a threshold-based alerting system. For instance, a threshold on the load metric wouldn't have helped, or anyway would have worked with some delay, and a uni-variate process wouldn't have been as good as a multi-variate approach.

4.2. Goals and Requirements

We already mentioned, during the previous chapters and sections, some of the constraints that our system must follow. Nevertheless, in order to have a complete list, all the functional, technological, and architectural requirements follow:

1. The system must be modular and it must be ready to adopt both the currently available and the new IT technologies used in the CERN infrastructure.
2. The implementation must proceed in a collaborative way with testing for every new feature.
3. The pre-processing step must be able to support the reading of big data using HDFS.
4. The system must be able to access several different input datasets and data sources.
5. The system must permit the adoption of the main ML libraries (like Sklearn, TensorFlow, PyOD, ...) both in notebooks and batch processing.
6. The code and the set of functions must allow “offline” explorative analyses and studies on pre-defined evaluation datasets.
7. The system must be able to scale and parallelize different tasks on the same data and models. It must be also possible to parallelize the jobs on different data and models.
8. The anomaly results published by the system must be accessible via the monitoring dashboards already used by the CERN cloud managers.
9. The implementation must include a job scheduling and monitoring system to automate the whole process.
10. Different parameters must allow to run the system in a testing environment, publishing anomaly results locally without being connected to the CERN central monitoring infrastructure.

11. The system must allow reproducible results for both the experiments and testing purposes.
12. The resources used by the system must be optimized in order to produce anomaly results using only a single VM.
13. The results must include metadata and information to be used in the monitoring framework.
14. The system's results dashboard must allow a fast consultation, highlighting the top anomalies between all the machines monitored by the system.

We will see in the next sections the design choices made by the previous work on the same topic, and the ones that we made to provide enough features with respect to these requirements.

4.3. Background

Our project, as mentioned in section 2.6, was not the first one trying to face the AD task in the CERN cloud infrastructure. A previous tech student already started to implement a library and a pipeline for this purpose [2]. A fast read of his thesis can help to understand the evolution of the project, but we can also summarize here what was done:

- The *Gitlab* repository related to the project was already created. Even if, looking at the tags, the branches, and the commits, the repository didn't evolve in a homogeneous manner, many features were already implemented. For instance, the support for *PyOD* and *TensorFlow* models was already present, and also functions to be called from bash commands were already there.
- The system was already implementing a first version of containerization using Docker.
- The first pipelines were already realized, including a first logic about checking if the needed data were already present.
- The anomalous scores were published using the CERN central monitoring infrastructure. Many bugs were affecting the results, but it is important to say that the communication between the system and the different frameworks was already implemented.
- Many notebooks were available. They were focusing on the first approach of the previous student, on understanding the problem and, especially, the kind of data obtainable from the cloud.

- A first labelled dataset was created. The dataset contained data about 2 different groups inside the cloud infrastructures, and it was created without cross-checking the labels. A very important point is that the situation of the cloud changed, and now there aren't anymore different groups in the CERN cloud but only the group of machines described in the previous chapters. Those machines now, only offer a heterogeneous collection of services and resources. Moreover, when creating a dataset, the absence of bias is fundamental and it can't be obtained without a cross-check of the labels.
- A comparison between many AD models was made using the labelled dataset for evaluation purposes. Even if the ML models were trained without any kind of hyperparameters optimization the study was a very good starting point giving us the possibility to focus on specific algorithms.

We must highlight the comparative study of the last point. That was a very important contribution and it allowed us to focus on the algorithms that were having the best results. In particular, this is the reason why we focused on improving the Isolation Forest and the autoencoders using Recurrent Short-Term units, which were the best models for, respectively, the traditional ML and the DL families. Moreover, some of our experiments proved the comparative study results, highlighting some issues and weaknesses of the algorithms having low performance.

Furthermore, the work mentioned in this section helped also for writing the first article about this system [1]. In the article we focus on the engineering part mentioning also the results regarding the Machine and Deep Learning models.

4.4. Design Decisions

Finally, we can explain our design decisions. The complete list of them follows.

- **(Req. 1):** For having a modular system we decided to use *Docker*². *Docker* is a set of Platform as a Service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Thanks to this ability, the system works using micro-services, which make easier the compatibility with different environments and enable the concept of encapsulation, always appreciate in the field of software engineering. Moreover, *Docker* is, since many years ago, the standard regarding micro-services and in particular, for our system, we have different Docker images for different purposes. For instance, the image containing our library, which updates

²<https://www.docker.com/>

itself after every new feature implementation, the image containing the pipeline orchestrator and scheduler, the image containing the tools for sending information to the database, and so on. All these Docker images, when running, “talk” to each other in order to achieve the final common goal.

- **(Req. 2):** Regarding the collaborative work, the standard versioning tool to use at CERN is *Gitlab*. This framework helps to organize the code, especially when we consider its evolution, history and different features. Moreover it offers many other options: for instance the automatic creation of Docker images or the possibility to run automatic CI/CD tests to be sure that the new code doesn’t break the functionality of the previous stable one.
- **(Req. 3):** We use *pySpark*³ for accessing and connecting to the CERN Spark Analytix Cluster⁴, for getting the data from *HDFS*, and for processing them. Once again, the decision is driven by the fact that people at CERN already use these kinds of tools and frameworks, and we have to adapt looking for the most compatible choice. Furthermore, *pySpark* offers many functions for dealing with big data objects.
- **(Req. 4):** For accessing different input datasets, we produced a generic implementation of the ETL phase (Extraction, Transformation, Load). Thanks to a declarative configuration file, used then by the ETL functions, we are able to transform and process different data coming from different specific sources.
- **(Req. 5, 11):** We wanted to be able to adopt the main know libraries regarding data science, data processing, machine and deep learning. In order to do it, we opted for a major usage of Python3⁵. We don’t have only Python in our project as a coding language, but it is anyway composing a big part of the repository. Moreover, with Python we are able to use our library⁶ in Notebooks⁷ for an interactive approach. Regarding the reproducibility, Python offers many ways to set random seeds for reproducible experiments and all the libraries that we use handle this aspect too.
- **(Req. 6):** As already mention in the section 1.3.2, we use SWAN for this kind of interactive approach we want to have. It is useful for exploring, in an offline mode, different solutions, and for preparing theoretical experiments.

³<https://spark.apache.org/docs/latest/api/python/>

⁴https://indico.cern.ch/event/716795/contributions/2946453/attachments/1645011/2628931/BigData_at_CERN_JP.pdf

⁵<https://www.python.org/>

⁶<https://gitlab.cern.ch/cloud-infrastructure/data-analytics/-/tree/qa-v0.4>

⁷<https://ipython.org/notebook.html>

- **(Req. 7, 9):** We built a pipeline, to be run every day, in order to compute and publish the anomaly scores. For running this pipeline, we use *Apache Airflow*⁸. Thanks to this tool, we are able to schedule the needed jobs in a parallelized manner and, especially, in an automatic way. Moreover, *Airflow* permits to monitor the scheduled processes, allowing us to see, for instance, the time needed for every task or the logs of every job. In addition, *Airflow* is easily deployed starting from a *Docker-Compose* configuration, and the tasks are designed via directed acyclic graphs (DAGs), implemented in Python code. In particular, we can have different DAGs running in the same moment implementing those desired features about scalability and parallelization.
- **(Req. 8, 14):** Thanks to *Fluentd* we are able to inject the anomaly results directly into the *ElasticSearch* infrastructure used by the cloud monitoring team at CERN. Once the related *Docker* container is running, *Fluentd* publishes to the desired infrastructure all the JSON files, with a specific defined structure, that we print. Having the data about the scores on *ElasticSearch* simplifies the monitoring of the scores themselves. For instance, we can use *Grafana* and create a dashboard to be consulted for visualizing the results.
- **(Req. 10):** We have different Docker images and containers for the production and for testing environments. Using bash scripts, with the possibility to indicate parameters at the moment of the run, we are able to change the *Docker-Compose* that is called, changing in this way the Docker containers underneath. For instance, saying to the system to run the testing environment we will use a Docker container connected to a local fake *ElasticSearch* service, instead of being connected to the official CERN one.
- **(Req. 12):** Optimizing the whole process, we are able to run our pipeline in a single VM, without the need of clusters or huge infrastructures. In particular, the VM we use in production is a “m2.2xlarge” instance having 16 virtual CPUs and 30 GB of RAM.
- **(Req. 13):** In the previous work [2], there was already a method for publishing the data with *Fluentd* with the right structure. We improved that method adding more important data related to the score, and refactoring what was already present.

⁸<https://airflow.apache.org/>

5 | Evaluation Dataset

After the description of the AD system, we move to the theoretical and experimental part of the thesis. We described the system first because it is the main output of our work, and because it is used on a daily basis in the unique context of the CERN cloud infrastructure.

Anyway, the selection of the models, the implementation of different mechanisms with respect to different models, and the changes in the preprocessing phase are all results of the experiments and the theoretical observations made during the period of this project. As we already mentioned in Section 1.4, one of the outputs is the labelled dataset, mandatory in order to evaluate the models with respect to certain metrics and for being able to make those observations previously mentioned.

Here we want to describe the dataset itself, why it is so important for us and for future works on this same topic, what changed with respect to the last labelled dataset used in [2], and how we proceeded with the creation of it in order to have the less possible bias.

5.1. Lack of Public Datasets

The first aspect to tackle is about the reasons behind the creation of a totally new labelled dataset. Usually, in many data science and ML projects, some public datasets are used to compare different approaches and models. In this case, the situation is different. Indeed, there are not many available datasets with data about multi-variate performance metrics regarding large-scale data centers. In particular, there are 2 main reasons behind this:

1. The first one is about the complexity of labelling such data. Indeed, we are talking about a huge amount of time series and data in general. Furthermore, anomalies are many times subjective and they change their definition depending on the specific use case. Creating a valid and not biased dataset is very challenging.
2. The second reason is about the companies that often have such big data centers. Our use case, having different types of cloud services offered to different types of users, is usually the same for commercial cloud providers or, in general, the same for big tech companies, which don't share labelled datasets about the metrics.

Now it is clear that, given those problems, and the uniqueness of our use case, we can't just use some random public datasets to evaluate the three algorithms. Moreover, if we decide to use public datasets, we are just repeating the work of others and we are using data very different from what we have at CERN.

5.2. Creation Process

It is worth describing briefly the creation process of our dataset. It is definitely not an easy task, especially if we consider the limited amount of time and the limited amount of people involved in the project. We had to think about a way to reduce the bias of the labels and about a way to make objective choices. Indeed, as we said, the subjectivity of identified anomalies is very high.

In Figure 5.1, we can see the first step of the process. Here we are individually labelling the hypervisors that are part of the subset we decided at the beginning to include in our dataset.

HOST	RangeStart	RangeEnd	AnomalyStart	AnomalyEnd	Comment	RangeLink	AnomalyLink
i73368050121962.cern.ch	1630447200000	1635721199000			NORMAL	link	link
i73368050132968.cern.ch	1630447200000	1635721199000	1635631140221	1635654422833	High Load, High ContextSwitches, Change of MemoryFree, Higher CPU	link	link
i73368050269093.cern.ch	1630447200000	1635721199000	1633530583572	1633534192864	Small Peak with higher load, very high Disk	link	link
i73368050296021.cern.ch	1630447200000	1635721199000	1632423245856	1632431655236	High Load, High ContextSwitches, High Disk	link	link
i73368050296021.cern.ch	1630447200000	1635721199000	1632484731900	1632811549866	Higher Load, Higher ContextSwitches, LONG PATTERN	link	link
i73368050466864.cern.ch	1630447200000	1635721199000	1630539217186	1630549114081	High Disk	link	link
i73368050466864.cern.ch	1630447200000	1635721199000	1631395947038	1631405724361	Higher Load, Higher ContextSwitches	link	link
i73368050466864.cern.ch	1630447200000	1635721199000			TOO MANY OTHER PEAKS ... LEAVING IT FOR NOW	link	link
i73368050711710.cern.ch	1630447200000	1635721199000			NORMAL	link	link
i73368051288767.cern.ch	1630447200000	1635721199000			LOL	link	link
i73368051349706.cern.ch	1630447200000	1635721199000	1630660126487	1631774335812	Load and Context always low, here in particular	link	link
i73368051501389.cern.ch	1630447200000	1635721199000			NORMAL	link	link
i73368051770359.cern.ch	1630447200000	1635721199000	1631019151168	1631065896601	High Load, High Context	link	link
i73368051770359.cern.ch	1630447200000	1635721199000	1634765059582	1634794279412	High Load, High Context, Disk Spike	link	link
i73368051770359.cern.ch	1630447200000	1635721199000			Maybe other peaks?	link	link
i73368052456345.cern.ch	1630447200000	1635721199000			NORMAL	link	link
i73368052798370.cern.ch	1630447200000	1635721199000			NORMAL	link	link
i73368052892711.cern.ch	1630447200000	1635721199000	1633414105849	1633418936128	Little Spike but of CPU, Disk, Load	link	link
i73368053404162.cern.ch	1630447200000	1635721199000	1630915467254	1630922343401	Spike Load, Disk	link	link
i73368053404162.cern.ch	1630447200000	1635721199000	1631063773235	1631072368418	Spike Load, Disk	link	link
i73368053404162.cern.ch	1630447200000	1635721199000	1633982004801	1634002611795	Spike Load, Disk	link	link

Figure 5.1: The first step of the dataset creation. Here we are labelling the host with a slightly biased approach, without comparing our decisions and without being affected by the others.

We can notice that, for every server, we are checking if it is normal for the all-time period under study. If it is the case, we simply add "NORMAL" as label. If it is not, we identify the intervals where we think there is an anomaly, describing it with a little comment. Furthermore, for every label, we have the link to the related monitoring dashboard in order to have a fast way to visualize the metrics in the specific interval we are labelling.

In Figure 5.2 we can see instead, the second step of the process. Here, after labelling the

anomalies independently, we are comparing the labels in order to make a final objective decision.

Host	Stiven	Antonin	Labels Antonin + comment	Labels Stiven + comment	ALL RANGE	Disagree
173368050121962.cern.ch	x	x	NORMAL	NORMAL	link	
173368050122968.cern.ch	x	x	1 LABEL	1 LABEL	link	
173368050269093.cern.ch	x	x	NORMAL	NORMAL	link	
173368050296021.cern.ch	x	x	NORMAL	NORMAL	link	
173368050466864.cern.ch	x-	x-	too many other anomaly spikes, lets not label	Too many peaks, OR ALL ANOMALOUS OR WE DONT LABEL	link	
173368050711710.cern.ch	x	x	NORMAL	NORMAL	link	
173368051288767.cern.ch	-	?	ANOMALY, but too many other anomaly spikes	Too many peaks and we can't say it is always anomalous. WE DONT LABEL	link	
173368051340706.cern.ch	x	x	1 LONG LABEL - Whole range extremely low Load+cs	1 LONG LABEL - Load and Context always low, here in particular	link	
173368051501389.cern.ch	x	x	NORMAL	NORMAL	link	
173368051770359.cern.ch	x-	x	1 LABEL	1 LABEL	link	
173368052456345.cern.ch	x	x	NORMAL	NORMAL	link	
173368052798370.cern.ch	x	x	NORMAL	NORMAL	link	
173368052892711.cern.ch	x	x	NORMAL	NORMAL	link	
173368053404162.cern.ch	x	x	7 LABELS - Load, memory ops, disk io, disk io_time	5 LABELS - Peaks of Load and Disk	link	
173368053830290.cern.ch	x	x	1 LABEL - Load, memory ops, disk io, disk io_time	1 LABEL - Single Peak of Load and Disk	link	
173368053965395.cern.ch	x	x	NORMAL	NORMAL	link	
173368054201405.cern.ch	x	x	1 LABEL - Higher load, extreme CS	1 LABEL - Single Peak of ContextSwitches	link	
173368054477502.cern.ch	x	x	NORMAL	NORMAL	link	
173368055028053.cern.ch	x	x	3 LABELS - Load, CS, Memory io, disk io_time	3 LABELS - Peaks of Everything (UNCERTAINTY!! THEY ARE ONLY 3?)	link	
173368055305677.cern.ch	x	x	NORMAL	NORMAL	link	

Figure 5.2: The second step of the dataset creation. Here, on the other hand, we are comparing the labels in order to remove the inevitable bias of the previous step.

The screenshot shows the final state of the sheet, but of course, while creating the dataset, this page changed many times. We were especially focusing here on intervals we labelled differently, and on intervals only present once. For each of these cases, we had to review our decisions, adding to the final dataset those intervals where, in the end, we were agreeing on, discarding, on the other hand, those where we still had doubts about.

5.3. Dataset Description

Let's finish the chapter by talking about the dataset itself. Indeed, it is important to know as much information as possible on the dataset, because all the metrics values in the evaluation part will be based on it.

In Figure 5.3, we can see an overview of the labels we created. The dataset contains data about 32 machines with respect to two months of data, from the 1st of September to the last day of October. In the heatmap, the blue intervals are related to normal intervals, the orange ones, on the opposite, refer to anomalous labels. In absolute values, we have a total of 11712 labels, 11484 normal and 228 anomalies. This means that we have the 2% of anomalous windows in our dataset.

This is the first nice property of the labels. The percentage of anomalies with respect to the number of normal data must be low, in order to have a context similar to reality. Moreover, if have too many anomalies, our starting assumption is not true anymore.

Finally, in Figure 5.4a and Figure 5.4b, we show two different examples of our labels. In the first case, as the caption mentions, we have an example where the anomaly is

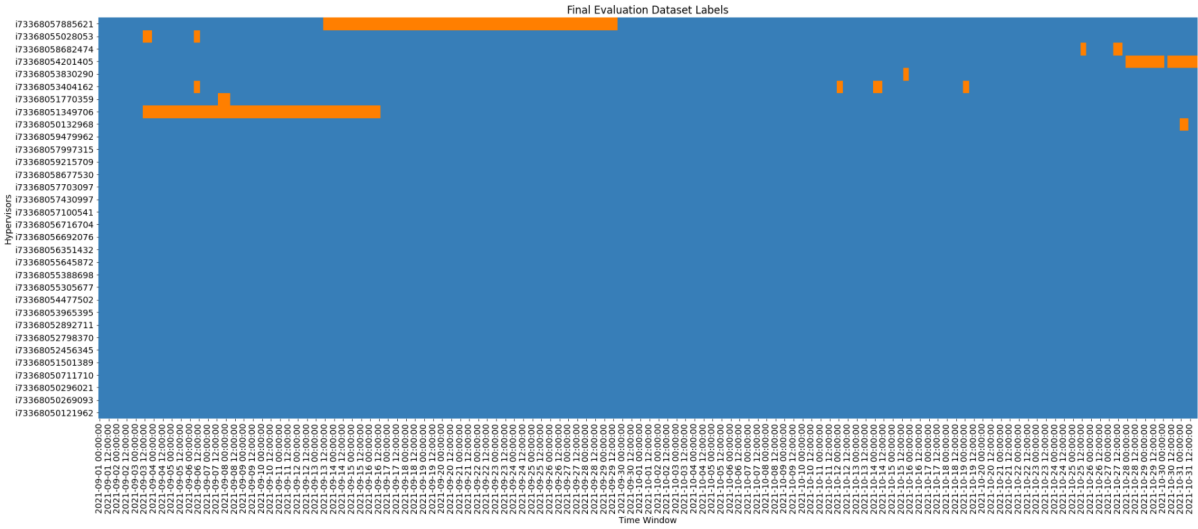
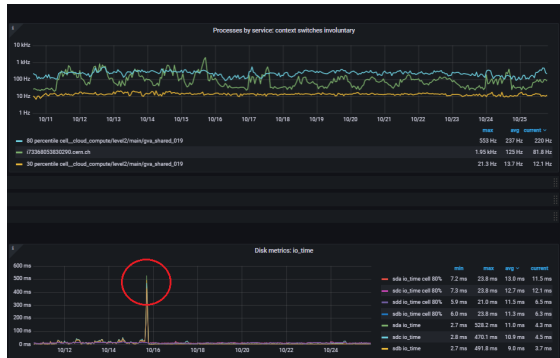
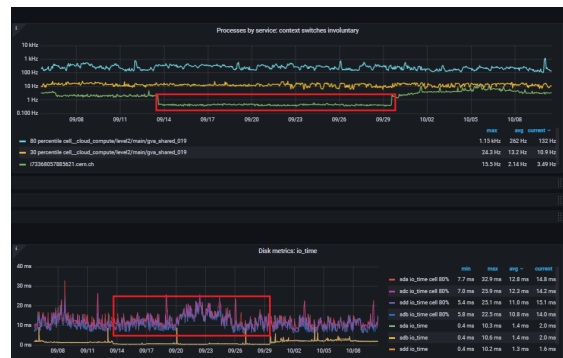


Figure 5.3: Heatmap of the labelled dataset. Each column represents a time interval, each row a specific hypervisor. Blue windows are normal, orange windows are anomalous.

very short, but it is not part of the noise. In particular, the anomaly doesn't last only some minutes but one or two hours, and for this reason, we want to be notified. In the second case, we have an anomaly long several days. Here, indeed, there is no spike, but a continuous misbehavior solved after a long time. Of course, there are many other examples talking about which anomalies are in the dataset, but here we wanted to show the 2 opposite cases with respect to the duration and values of the metrics.



(a) Example of an anomalous "spike". Notice that a spike in this case is anyway related to a relevant period of time with respect to our use case.



(b) Example of a longer anomaly. Remember that, in our assumptions, we consider also this case as anomalous

Figure 5.4: Examples of labelled anomalies.

6 | Experiments

In this chapter, we will present the experiments we did using our labelled dataset, talking also about some related implementation details. Furthermore, we will list the results of those experiments and we will discuss them, trying to get insights about the ML models we are using and about the AD system itself. In particular, we will show the results about the comparison between our system and the previous one used for spotting anomalies.

6.1. Figures of Merit

First of all, in order to understand all the experiments and, especially, the related results, we have to introduce all the evaluation metrics we will use.

6.1.1. Precision, Recall, F1-Score

In our problem, as defined in section 3.1, we want to classify metric windows as normal or as anomalous. This is a typical classification ML problem, in particular, it is a binary problem, because of having two specific classes. For these kinds of problems, we usually refer to the confusion matrix, shown in Table 6.1, for understanding which performance metrics to use for the evaluation.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Table 6.1: The confusion matrix of a binary problem.

For each window of time we have in input, given the true class and the predicted one, we have that the instance is a:

- **True Positive**, if the window is anomalous (positive) and we predict an anomaly,
- **False Negative**, if the window is anomalous (positive) and we predict it as normal,

- **False Positive**, if the window is normal (negative) and we predict an anomaly,
- **True Negative**, if the window is normal (negative) and we predict it as normal.

We call TP the total number of True Positives in our dataset for a certain experiment. The same logic is applied to FN, FP and TN. Given these four quantities, we are able to obtain many performance metrics. The usually most used one is the accuracy, defined in 6.1, which compares the true labels and the total number of labels.

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \quad (6.1)$$

The accuracy is very intuitive and it often allows us to see in a fast way the results of an experiment. In our case anyway, we have an unbalanced problem: let's remember that our general assumption is to have very few anomalies with respect to the normal data, and in particular, as mentioned in Section 5.3, we must remember that we have the 2% of anomalies in our dataset. The accuracy, unfortunately, doesn't help in this case: the fewer anomalies we have, the more accuracy I have, despite the performance of the model we want to test. We have to use other two metrics:

- the precision, defined in 6.2, which measures how many true positives we have between all the predicted positives, and

$$precision = \frac{TP}{TP + FP} \quad (6.2)$$

- the recall, defined in 6.3, which measures how many true positives we have between all the real positives.

$$recall = \frac{TP}{TP + FN} \quad (6.3)$$

If we want to minimize the False Positives, we try to maximize the precision, finding a balance anyway between it and the recall. On the contrary, if we want to minimize the False Negatives, we go for higher recall. There is, however, a third metric that can be used to compare different models: the f1-score.

The f1-score, defined in 6.4, combines the previous two metrics. It gives the same importance, more or less, to False Negatives and False Positives, allowing to have an unbiased evaluation that considers different use-case.

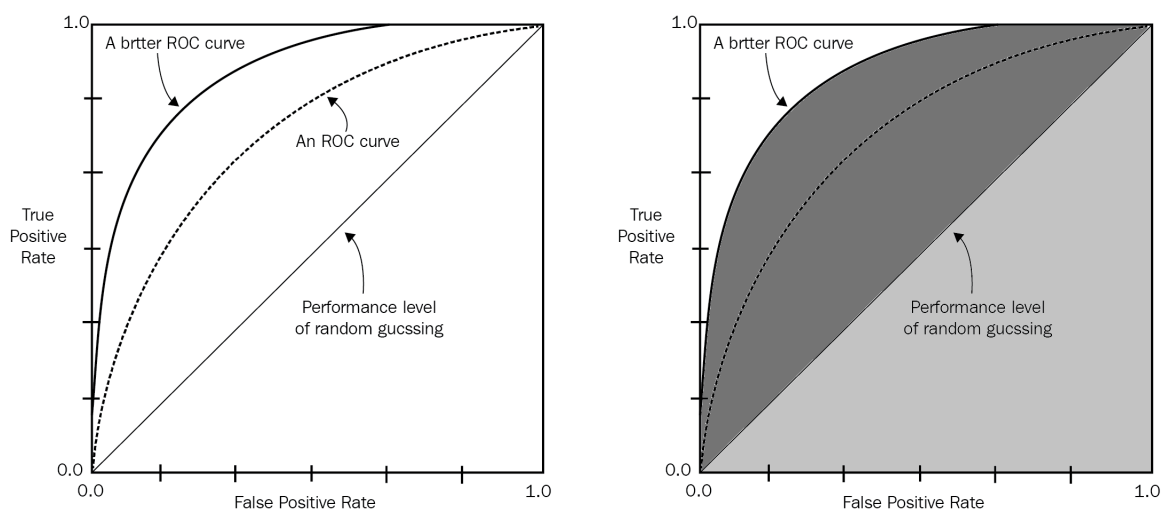
$$f1-score = 2 \times \frac{precision \times recall}{precision + recall} \quad (6.4)$$

6.1.2. Receiver Operating Characteristic and AUC-ROC

The major problem when using these metrics is that we need to predict the final binary class for computing them, and this means being dependent on the threshold Γ defined in the function 3.7. Because of this, we will often evaluate our models using the Receiver Operating Characteristic (ROC) curve, shown in Figure 6.1a. The curve is created by plotting the True Positive Rate (TPR), called also “sensitivity” and equal to the recall, against the False Positive Rate (FPR), called also “probability of false alarm”, at various threshold settings.

Thanks to the threshold independence, we are able to compare the models between them, to see how the models perform with respect to random guessing, and given a FPR value, to compare the True Positive Rate of specific models.

Furthermore, we can collapse those details and information into a single number in order to get a fast response and to visualize the results of larger experiments (we can’t really visualize hundreds of plots, but we can visualize the mean and the standard deviation of a certain number summarizing the performance). This number is the Area Under the Curve (AUC), shown in Figure 6.1b. It literally represents how large is the area under the ROC curve, and for this reason, we also refer to the AUC-ROC metric. The maximum value is 1, and the random guessing has an AUC-ROC equal to 0.5.



(a) The Receiver Operating Characteristic curve¹. The closer to the top left corner, the better.

(b) The Area Under the Curve. A better curve implies a larger area.

Figure 6.1: ROC curves examples.

¹<https://www.oreilly.com/library/view/neural-networks-with/9781788397872>

6.2. Experimental Setup

Let's briefly describe the setup used for the experiments. This is relevant in order to know exactly how the experiments are run and to judge the results themselves. In particular, the time performance are strictly correlated to the characteristics of the machines we use.

As already mentioned in section 1.3.2, we run the experiments in an interactive environment to be able to modify our library on-the-fly. This interactive environment, in particular, is called SWAN [41] and it allows to perform interactive data analysis in the cloud, to write and run data analyses leveraging on the Jupyter notebook interface.

In Figure 6.2a we have the configuration setup offered by SWAN when accessing the service: in our case, we use a *Docker* container running CentOS 7, with 4 Intel(R) Xeon(R) E5-2630 v3 2.40GHz, 16 GB of memory and a connection to the Analytix² cluster offering tools like Spark and HDFS.

In Figure 6.2b, on the other hand, we have a subset of the notebooks produced during the thesis work. Some notebooks are about data inspection or data processing, some of them are about running experiments and visualizing the results, and others are about showing to other teams our work or about debugging issues.

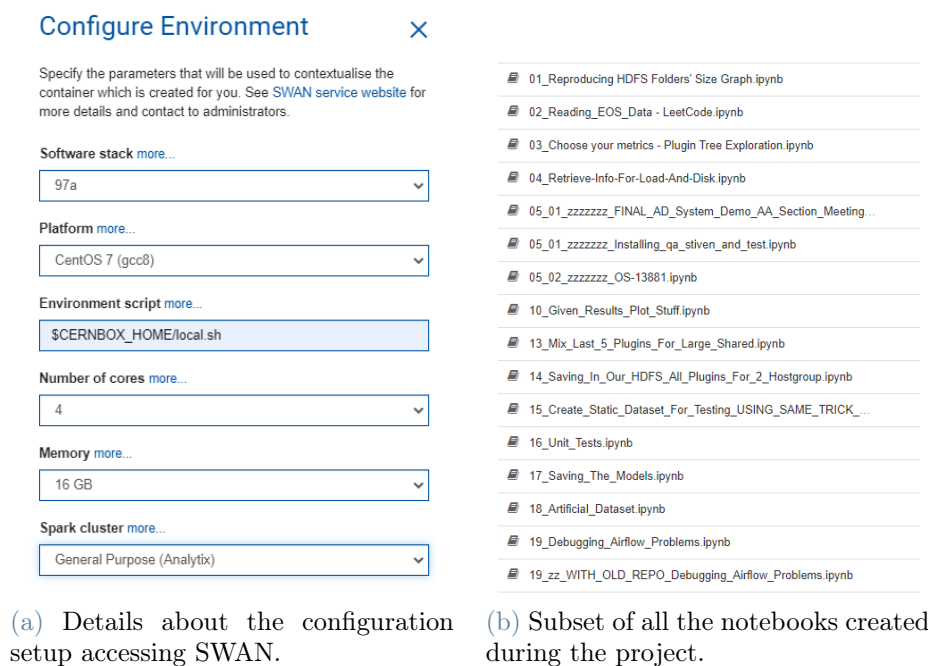


Figure 6.2: SWAN details.

²https://indico.cern.ch/event/716795/contributions/2946453/attachments/1645011/2628931/BigData_at_CERN_JP.pdf

6.2.1. Parameters Selection

In our problem, we have 2 main different types of parameters: the ones directly related to the anomalies and the ones related to the models' architectures. Regarding the first family of parameters, we should not select them in order to improve the performance, but, on the opposite, we have to decide those even before starting the experiments. For deciding their values we have to consider the experts' opinions and the type of anomalies that we want to detect.

Regarding the second type, we want instead to improve generally the performances and the stability of the models. Unfortunately, however, we can't really perform any pragmatic hyperparameter optimization, because of the unsupervised approach of our use case. Here we can just use empirical early results and experts feedback, which can give us some hints about the fair values of the parameters.

In Table 6.2 we list the main parameters of the system and their values. For the following experiments, we will use these parameters, unless the experiment is about a specific one between them. In that case, we will keep the others at the same values shown here, and we will change the parameter under study.

Parameter	Value	Motivation
Aggregation Resolution	30 Minutes	In order to avoid punctual outliers. Parameter from the experts.
Window Size	8	We are interested in detecting anomalies in the order of 4 hours. Parameter from the experts.
Training Set Size	4 Weeks	To feed the models with the right amount of data. Empirical parameter.
Number of iTrees	200	To be sure the model performance are stable. Empirical parameter.
LSTM/GRU Units	1	A less complex architecture doesn't overfit the data. Empirical parameter.
Epochs	50	To be sure to not underfit the data. Empirical parameter.
Dropout Rate	0.25	Dropout layers are added for generalization of the architecture. Empirical parameter.

Table 6.2: The values of the main parameters of our system. We can see, for each parameter, the value we set and the motivation. Notice that the first 3 parameters of the table are common to all the models, while the "Number of iTrees" is only related to the Isolation Forest and the last 3 parameters are only related to the autoencoders.

6.3. Results

We will present here the experiments that we approached during the work of the thesis, and their related results. Notice that you will find the description of the experiments, their motivation and the details about the results. We will not try to extrapolate information from those results, but will “just” present them, leaving the discussion to the next section.

6.3.1. Performance of the Proposed Solution

The first experiments are about asserting the performance of the three models with respect to the evaluation dataset. In particular, we will measure the AUC-ROC metric for each of them in order to

- have a comprehensive comparison between each other,
- know the average performance of the models, and
- find the models behavior with respect to some parameters.

AUC-ROC of Individual Models

We explore how the model performance changes when varying the temporal interval of the training set and/or the initialization seed. In particular, for each model, we execute a total of 45 independent runs of model training by using 15 non-overlapping, one week-long, training sets and three initialization seeds.

Model	AUC-ROC Week	AUC-ROC Month
IFOR	0.954 ± 0.006	0.959 ± 0.004
LSTM-AE	0.964 ± 0.010	0.964 ± 0.010
GRU-AE	0.962 ± 0.010	0.971 ± 0.005

Table 6.3: Stability of model performance, assessed by the average and standard deviation of the AUC-ROC, when varying the time interval of one week-long (AUC-ROC week) and one month-long (AUC-ROC Month), training sets.

For each model, the average and standard deviation of the AUC-ROC values computed over the 45 experiments are reported in Table 6.3 (‘AUC-ROC Week’ column). The table reports also the same statistics obtained from other 12 experiments using 4 non-overlapping, one month-long, sets of data and the three initialization seeds (AUC-ROC

Month' column). The average AUC-ROC value of all models is above 0.95, with a standard deviation extremely small, being less than 1 % of the average value.

AUC-ROC when Changing Parameters

We want to analyze the performance of the three models when changing some of the parameters characterizing their behavior. We start with IFOR. In Figure 6.3, we show the AUC-ROC metric of IFOR when changing the parameter related to the number of iTrees it uses. Also here we are changing seed and doing multiple experiments to compute the standard deviation and plot the error intervals.

We can see that for a low number of iTrees the model is not robust and, indeed, the variance is very high. When the number of iTrees increases, we have a consequent increase in the AUC-ROC values. Moreover, also the standard deviation decreases. In particular, we have a stable AUC-ROC with more than 30 iTrees.

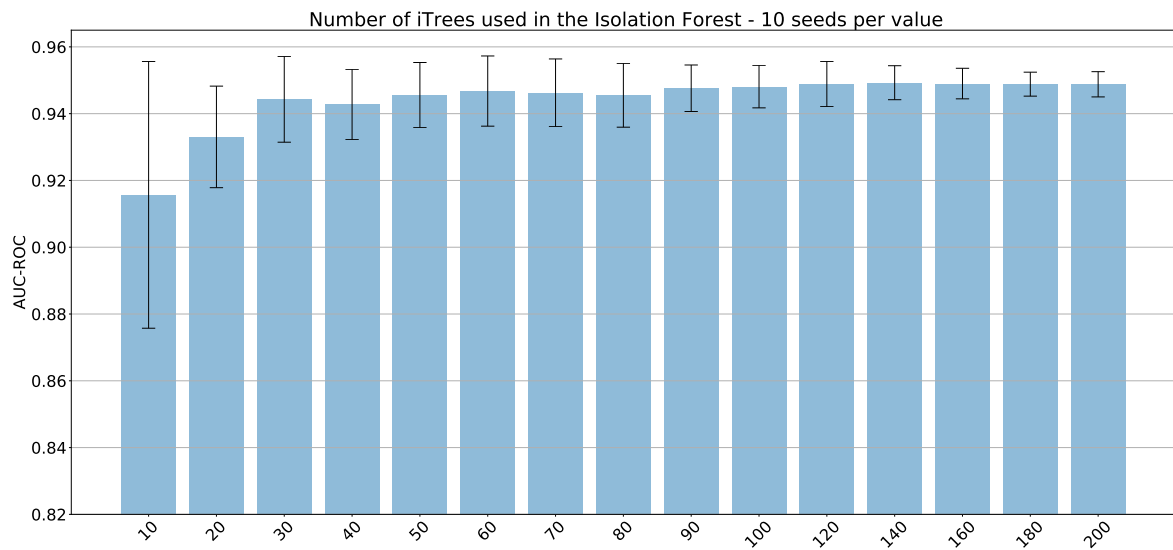


Figure 6.3: The AUC-ROC metric of the Isolation Forest when changing number of iTrees the model creates and uses.

Another aspect we investigate is the dependency of the LSTM/GRU autoencoders' AUC-ROC value from some architecture variants. In particular we explore the effect of applying dropout and varying the number of recurrent units, which are two strategies to prevent over-fitting during the training phase. The effect of introducing a layer dropout of 25% has been combined with the usage of a different number (one or three) of LSTM/GRU recurrent units in the encoding and decoding layers of the models. For each configuration, we have performed 12 experiments, using four different one week-long training sets and

three different initialization seeds. Table 6.4 reports the average and standard deviation of the AUC-ROC values computed over all experiments. The highest average AUC-ROC values are obtained for both models when dropout is used. The GRU-AE looks to benefit more from dropout, with not only an higher average value but also a smaller result variability, as shown by the standard deviation values. On the contrary, the adoption of one or three units does not seem to bring much benefit.

Parameters	AUC-ROC LSTM	AUC-ROC GRU
No Dropout, 1 Unit	0.92 ± 0.05	0.86 ± 0.10
No Dropout, 3 Units	0.90 ± 0.03	0.80 ± 0.10
25% Dropout, 1 Unit	0.970 ± 0.004	0.97 ± 0.01
25% Dropout, 3 Units	0.96 ± 0.02	0.96 ± 0.01

Table 6.4: Performance of the proposed LSTM/GRU autoencoder models when including 25% layer dropout and/or three units. The performance is measured by the average and standard deviation of the AUC-ROC values computed over 12 experiments with different, one week-long (AUC-ROC week), training sets.

6.3.2. Comparison with Previous System

The main goal of this thesis is to show that our anomaly detection solution outperforms the previous system implemented in the CERN cloud for spotting anomalous servers. To verify that this is the case, we compare the performance of our (individual and ensemble) models with respect to the performance of the previous system. We guarantee a fair comparison by configuring the previous system to analyze each time series in four hours long windows and to consider the same performance metrics of our solution. We manually adjust the starting thresholds of the previous system, to achieve different target values of FPR in the following experiments. The performance comparison is expressed in terms of the TPR values at a given value of FPR.

In Table 6.5 the TPR values are reported for all the approaches, evaluated at four different values of FPR, ranging from 0.1% to 4%. Due to large number of servers in the cloud infrastructure, a small FPR is mandatory to avoid that false notifications submerge the service managers. The results show that our solutions systematically outperform the previous system. The performance of IFOR at the lowest FPR are comparable with the one of the previous system, whereas, at 4% of FPR, they become similar to the LSTM-AE's ones. Between the individual models, the autoencoders have the best performance,

with GRU-AE achieving higher TPR for higher values of FPR.

Regarding the ensembles, ENS_1 and ENS_2 give the best performance at the lowest FPR, where the inefficacy of IFOR penalises ENS_3 . However, the three ensemble methods have similar performance at higher values of considered FPR.

FPR	True Positive Rates						
	Previous System	Individual			Ensemble		
		IFOR	LSTM-AE	GRU-AE	ENS_1	ENS_2	ENS_3
0.001	0.08	0.09	0.45	0.43	0.47	0.45	0.21
0.01	0.14	0.29	0.56	0.58	0.53	0.58	0.59
0.02	0.19	0.57	0.66	0.79	0.61	0.69	0.71
0.04	0.26	0.81	0.81	0.93	0.92	0.89	0.92

Table 6.5: True positive rate (TPR) measured at different values of the False Positive Rate (FPR) for the predictions of the previous AD system, the proposed individual models and the proposed ensemble strategies ENS_e for $e = 1, \dots, 3$.

6.3.3. Minor Results

In Appendix A, we find other two figures that can bring interesting results:

- In Figure A.1, we can see the loss function behavior related to the training of the two autoencoders. In particular, we distinguish between the use of one week and one month of training set. In the first case, we can notice that the loss takes more epochs to decrease and that the saturation value is equal to 0.008. On the opposite, in the second case, we have that the loss takes fewer epochs to reach the minimum.
- In Figure A.2, we can see, instead, the ROC curves and the score distributions of the three models. In particular, is not really possible to show all the experiments, so we decided to have a general overview showing the plots in the best and in the worst cases. In particular, the best case is related to those situations where the models perform well, while the worst case is related to the use of fewer data and worse parameters. Notice that, the model performs well if it is able to separate the two classes in the score distribution..

We also present here the results related other minor experiments. In particular, we study the difference between using non-overlapping or overlapping metric windows, we inspect the importance of the number of dimensions used in the windows, and we show the time

performance of the models.

Window Sliding

Our definition of anomaly implies using the temporal dimension, and our solution involves the definition of metric windows to be classified as anomalous or normal. As shown in Figure 6.4, one technique for enlarging the training dataset, and for feeding with more samples the models, consists in creating overlapping windows using a sliding approach.

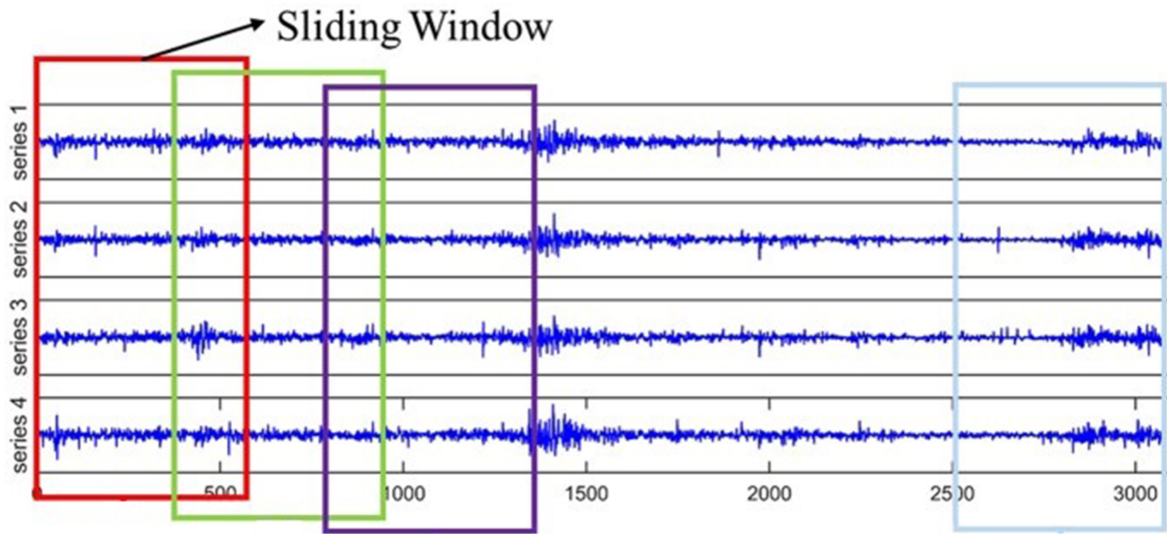


Figure 6.4: A visualization of the sliding window technique^[42]. Overlapping the windows we are able to increase the training dataset size.

In these experiments, we wanted to study the performance in the case of using or not this sliding approach. This technique, indeed, has some negative aspects that can decrease the models' efficiency. For instance, if anomalous data are present in the starting training set, with sliding windows, we propagate those anomalies on more samples.

The related experiment is shown in Figure 6.5. We tried to apply the sliding window approach with respect to two specific metrics (*CPU Load* and *Disk IO Time*), increasing the time period involved in the training set, and for this reason, increasing the number of data points in the training set itself.

The plots show the drawback of this approach. Being the metrics' values close in time, they are also close in absolute value. Increasing the number of data points we find that artificial clusters of data points are created. This implies that an anomalous data point will be part of a cluster if we use this approach, and of course, we want to avoid this situation.

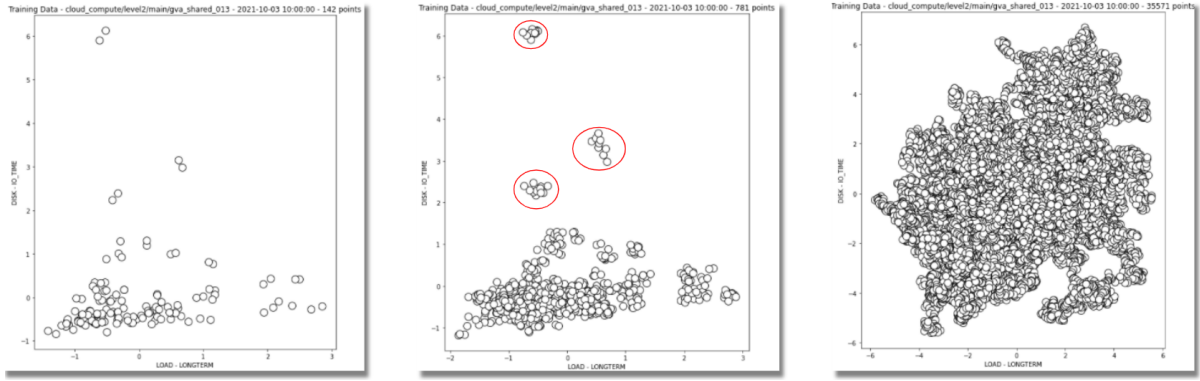


Figure 6.5: The plots show the context of the experiment about sliding windows. We are plotting two metrics and we are increasing the number of data points.

Another option is to explain the temporal information of overlapping windows adding, instead of data points, more dimensions. It is actually what we do when we define the metric window $W_{i,k}$ in (3.5). We wanted however to visualize what happens if we abuse of this approach. In Figure 6.6 we can visualize a summary of the experiment. Notice that, as said in the caption, adding dimensions we had to reduce the dimensionality by using the Principal Component Analysis for plotting the data. What we can see is that, in the right final plot, the data points are really far from each other because of the “Curse of Dimensionality” [43]. This means that, when running AD models, we will not be able to identify clusters containing normal data points or distant anomalous points.

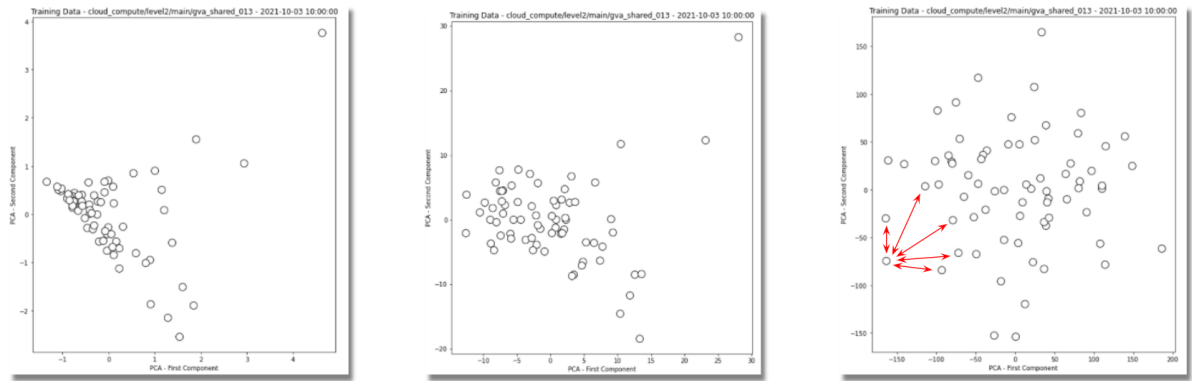


Figure 6.6: The plots show the context of a second experiment about increasing the amount of data. We are plotting the same metrics of Figure 6.5, but in this case, we increase the number of dimensions instead of increasing the number of data points. Notice that, because we are increasing the number of dimensions, we have to apply a dimensionality reduction for visualizing the data points. In particular, we use the PCA [30].

Time Performance

Finally, we want to establish the time performance of the three models. We already know that the autoencoders take more time for the training phase with respect to the Isolation Forest, given the same amount of data. We want, anyway, to understand the absolute value of this gap between the two families of models, to measure the time needed also for the inference phase, and to compare the LSTM and the GRU units used in the two DL models. Especially in our case study, the time performance is a key aspect for deciding which model select, and for understanding the capabilities of the different approaches.

In Figure 6.7, we plot the training and inference time that IFOR needs, in function of the number of iTrees used. We can observe that the absolute value of the training is really low. Most importantly, the time increases linearly with respect to the number of iTrees for both the training and the inference. Notice that the training data is composed of one week of data, while the inference dataset is related to two months.

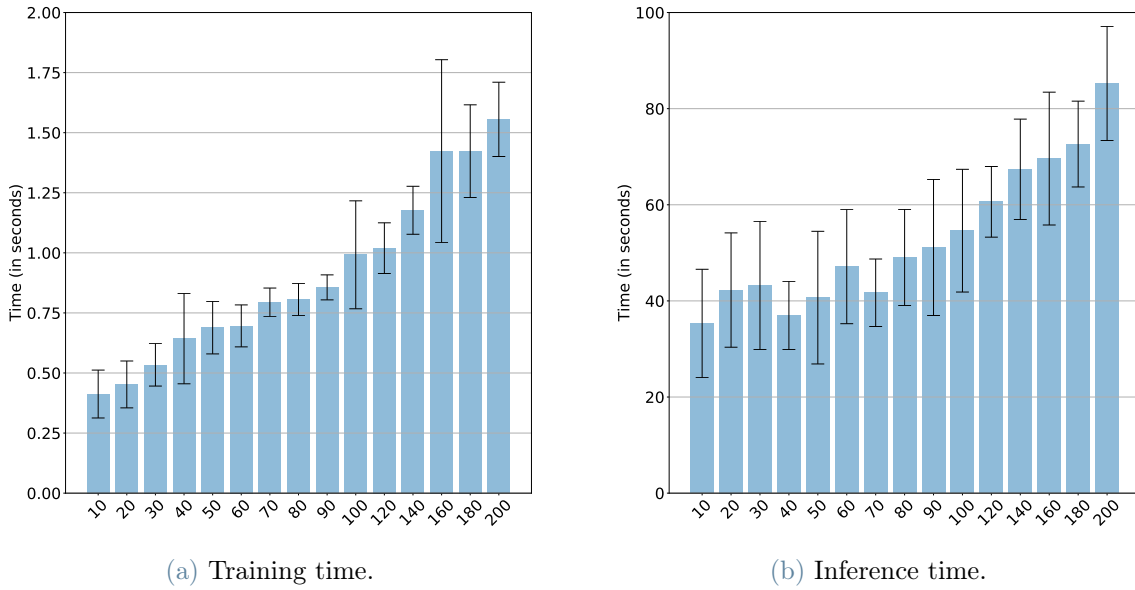


Figure 6.7: The time performance of Isolation Forest when increasing the number of iTrees the model creates and uses.

In Table 6.6 we can see the time that each of the models we are considering needs for both the training and inference phases. In particular, we can observe that, as expected, IFOR needs almost no time for its training phase. On the contrary, with respect to the inference phase, the three models have similar values.

Model	Training Time	Inference Time
IFOR	1.63 ± 0.29	94.93 ± 16.83
LSTM-AE	64.5 ± 9.61	107.8 ± 19.36
GRU-AE	83.1 ± 9.83	100.1 ± 22.81

Table 6.6: Training and Inference time, in seconds, of the three models. The training time is related to one week of data, the inference time is related to two months.

6.4. Discussion

In this last section, we want to retrieve the important information that we can get from the results of the previous section, and comment on them in order to extrapolate the main messages. Given those results, we list all the considerations:

- The first consideration that we can make is about the different nature of the models and their similar performance. Indeed, we see in Table 6.3 that the three models have all $\text{AUC-ROC} > 95\%$. However, IFOR is a traditional ML algorithm that focuses on the statistics of the data, while the autoencoders, even if they need to be properly trained, build a non-linear model able to reproduce correctly the normal inputs. This difference is highlighted in Table 6.5, when we see that IFOR is not able to have a high TPR for a very low given FPR.
- Secondly, we proved the robustness of the IFOR. In almost all the results that we presented, IFOR is always the model with less variance in the performance.
- The next consideration is, on the other hand, about the two autoencoders we are considering. They perform in general slightly better than IFOR, and, especially with more data in the training set, they are also robust. Furthermore, using dropout layers in the architecture seems crucial to have good performance, and it seems that using one or multiple LSTM/GRU units doesn't matter. Probably, our use case doesn't need a complex model to be tackled, and using no dropout makes the autoencoders prone to overfit.
- Finally, with those experiments we found out that our new proposed approach outperforms the previous system. In particular, if we set the required False Positive Rate to 0.001 we have an improvement of the True Positive Rate of 39% reaching the 47%. In this way, we are able to get almost half of the true anomalies, paying the fact of finding 1 False Positive every 1000 cases.

7 | Conclusions and Future Work

Large-scale cloud computing infrastructures require automatic detection tools for reporting any server following an anomalous behavior. Solving this problem by machine learning models is very challenging due to the large number of time series monitoring metrics being steadily acquired, the large variability of those in normal conditions and the scarcity of annotations.

In this thesis, we presented an unsupervised anomaly detection approach suitable for multi-variate time series acquired in the large-scale cloud computing infrastructure of the European Organization for Nuclear Research (CERN). Our working hypothesis relies on the observation that the multitude of computer servers in a data center are generally organised in clusters. Furthermore, servers in the same cluster show, during their standard operation, a similar average behavior, whereas a handful of them can manifest temporary deviations defined as anomalous behaviors.

The proposed solution is completely unsupervised, and employs ML models trained to simultaneously describe all the metrics acquired from servers belonging to the same cluster. These time series are arranged in a multivariate time series and monitored by three different detectors, namely Isolation Forest, LSTM-autoencoder and GRU-autoencoder, which are combined in an ensemble.

7.1. Outputs and Contributions

We summarize here the main contributions and outputs of our work (including points mentioned in section 1.4). Firstly, we developed an Anomaly Detection Python library able to perform such task in a parametric fashion for different use cases. Then, we implemented a system, which is running every day since the beginning of 2022, able to automatically detect anomalous servers in the CERN Cloud Infrastructure. In Figure 7.1, it is possible to see an example of an anomalous server detected by it during the daily operations. We also created a representative and unbiased labelled dataset that can be used to evaluate future models. Furthermore, we showed that, in terms of AUC-ROC,

the three adopted models achieve high performance ($\text{AUC-ROC} > 0.95$), and that they are independent to the training data selection and size.



Figure 7.1: The anomalies detected by our system in production. Notice that every red window is a 4 hours interval detected as anomalous by the models.

Finally, our experiments demonstrated that the proposed solution outperforms by far the threshold-based system previously employed at CERN, achieving a true positive rate of 47% against 8% of the previous system, when configured to yield 0.1% false positive rate. This evidence indicates that relevant anomalies need to be detected by analyzing patterns and structure of time series, and that is not sufficient to assess whether these contain outlying values.

7.2. Future Work

Possible future work can bring improvements to the system that we developed. In particular, for our approach, we implemented IFOR, LSTM-AE and GRU-AE. Those models are already implemented in many libraries and they don't have a specific architecture for tackling our problem. Building a completely new model with a peculiar architecture can be interesting and can bring novelty to this use case. Moreover, a deeper study can be done on the input data. For instance, different pre-processing steps can be tried, or more metrics can be added in the system input, looking then at the performance that those changes bring.

Another possible future contribution is related to the dataset. We already mentioned in

section 5.1 the challenges of labelling the time series under study. For this reason, an extension of the labelled dataset, with the addition of more servers can really improve the accuracy of the metrics used to evaluate the models and the different approaches.

Finally, a recent line of research is working with application logs [44–48]. These contain much more information and sometimes can also directly explain the cause of detected anomalies. This information unfortunately is mostly unstructured, presenting many not trivial challenges, and it was no intention of this project to approach those kinds of inputs. Nevertheless, future work on the logs of the machines in the CERN cloud infrastructure could be attractive, with a potential improvement with respect to the system used now by the CERN cloud managers.

Bibliography

- [1] Domenico Giordano, Matteo Paltenghi, Stiven Metaj, and Antonin Dvorak. Anomaly detection in the cern cloud infrastructure. *EPJ Web of Conferences*, 251:02011, 01 2021.
- [2] Matteo Paltenghi. Time Series Anomaly Detection for CERN Large-Scale Computing Infrastructure. Oct 2020. Presented 02 Oct 2020.
- [3] Qi Zhang, Lu Cheng, and R. Boutaba. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 05 2010.
- [4] Travis Brummett, PJ Sheini, Debadrita Sarkar, and Michael Galloway. Performance metrics of local cloud computing architectures. 11 2015.
- [5] Mohammad Sadegh Aslanpour, Sukhpal Singh Gill, and Adel Toosi. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things*, 12:100273, 08 2020.
- [6] Zheng (Eddie) Li, Liam O’Brien, He Zhang, and Rainbow Cai. On a catalogue of metrics for evaluating commercial cloud services. 02 2012.
- [7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41, 07 2009.
- [8] Xiaodan Xu, Huawen Liu, and Minghai Yao. Recent progress of anomaly detection. *Complexity*, 2019:1–11, 01 2019.
- [9] Mausumi Das Nath and Tapalina Bhattasali. Anomaly detection using machine learning approaches. *Azerbaijan Journal of High Performance Computing*, 3:196–206, 12 2020.
- [10] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection. *ACM Computing Surveys*, 54(2):1–38, Apr 2021.
- [11] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.
- [12] Ryan Taylor, Frank Berghaus, Franco Brasolin, Cristovao Cordeiro, Ron Desmarais,

- Laurence Field, Ian Gable, Domenico Giordano, Alessandro Girolamo, John Hover, Mirka Leblanc, Peter Love, Michael Paterson, Randall Sobie, and A. Zaytsev. The evolution of cloud computing in atlas. *Journal of Physics: Conference Series*, 664:022038, 12 2015.
- [13] José León. Advanced features of the cern openstack cloud. *EPJ Web of Conferences*, 214:07026, 01 2019.
- [14] T Bell, B Bompastor, S Bukowiec, J Leon, M Denis, J Eldik, Marcos Fermin Lobo, L Alvarez, Daniel Fernandez Rodriguez, A Marino, B Moreira, B Noel, T Oulevey, Wataru Takase, A Wiebalck, and S Zilli. Scaling the cern openstack cloud. *Journal of Physics: Conference Series*, 664:022003, 12 2015.
- [15] G Aad, E Abat, Jasmin Abdallah, Ahmed Abdelalim, Abdesselam Abdelouahab, O Abdinov, Wafa Amatullah, M Abolins, H Abramowicz, E Acerbi, Basanta Acharya, Ricardo Achenbach, M Ackers, D Adams, F Adamyan, Tetteh Addy, M Aderholz, C Adorisio, Paolo Adragna, and V Zychacek. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3:S08003, 08 2008.
- [16] S. Chatrchyan, Gevorg Hmayakyan, V. Khachatryan, Jehad Mousa, W Adam, T. Bauer, Thomas Bergauer, H Bergauer, Marko Dragicevic, J. Erö, and Ryszard Romaniuk. The cms experiment at the cern lhc. *Journal of Instrumentation*, 3:S08004, 08 2008.
- [17] The Collaboration, Augusto Alves, L Filho, A. Barbosa, Ignacio Bediaga, G. Cernicchiaro, G. Guerrier, Ana Machado, J. Magnin, F Marujo, J. Miranda, Andréa Reis, A Santos, A Toledo, Kazuyoshi Akiba, S Amato, B. Paula, Leandro de Paula, and T Ypsilantis. The lhcb detector at the lhc. 01 2008.
- [18] Luis Alvarez, Olga Datskova, Ben Jones, and Gavin McCance. Managing the cern batch system with kubernetes. *EPJ Web of Conferences*, 245:07048, 01 2020.
- [19] Alberto Aimar, Asier Corman, Pedro Andrade, Javier Fernandez, Borja Bear, Edward Karavakis, Dominik Kulikowski, and Luca Magnoni. Monit: Monitoring the cern data centres and the wlcg infrastructure. *EPJ Web of Conferences*, 214:08031, 01 2019.
- [20] Mingyan Teng. Anomaly detection on time series. In *2010 IEEE International Conference on Progress in Informatics and Computing*, volume 1, pages 603–608, 2010.
- [21] Oleksandr I. Provotar, Yaroslav M. Linder, and Maksym M. Veres. Unsupervised

- anomaly detection in time series using lstm-based autoencoders. In *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, pages 513–517, 2019.
- [22] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.
- [23] Andrew A. Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020.
- [24] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. pages 413 – 422, 01 2009.
- [25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [26] Rahul Dey and Fathi Salem. Gate-variants of gated recurrent unit (gru) neural networks. 01 2017.
- [27] Zhiguo Ding and Minrui Fei. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. In *ICONS*, 2013.
- [28] Mohammad Braei and Sebastian Wagner. Anomaly detection in univariate time-series: A survey on the state-of-the-art. *CoRR*, abs/2004.00433, 2020.
- [29] Wentai Wu, Ligang He, Weiwei Lin, Yi Su, Yuhua Cui, Carsten Maple, and Stephen A. Jarvis. Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.
- [30] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.
- [31] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2725–2732. International Joint Conferences on Artificial Intelligence Organization, 7 2019.
- [32] Y. Guo, Weixian Liao, Qianlong Wang, Lixing Yu, Tianxi Ji, and Pan Li. Multidi-

- mensional time series anomaly detection: A gru-based gaussian mixture variational autoencoder approach. In *ACML*, 2018.
- [33] Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur, Vanish Talwar, Wade Satterfield, and Karsten Schwan. Statistical techniques for online anomaly detection in data centers. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 385–392, May 2011. ISSN: 1573-0077.
- [34] Mohiuddin Solaimani, Mohammed Iftexhar, Latifur Khan, Bhavani Thuraisingham, and Joey Burton Ingram. Spark-based anomaly detection over multi-source VMware performance data in real-time. In *2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 1–8, December 2014.
- [35] Rodrigo N. Calheiros, Kotagiri Ramamohanarao, Rajkumar Buyya, Christopher Leckie, and Steve Versteeg. On the effectiveness of isolation-based anomaly detection in cloud data centers. *Concurrency and Computation: Practice and Experience*, 29(18):e4169, 2017. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4169>.
- [36] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Georges Kaddoum, Albert Y. Zomaya, and Rajiv Ranjan. A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks. *IEEE Transactions on Network and Service Management*, 16(3):924–935, September 2019. Conference Name: IEEE Transactions on Network and Service Management.
- [37] Henry Leung and Siyue Chen. Anomaly detection for cloud monitoring, October 2014.
- [38] Ang Li, Lin Gu, and Kuai Xu. Fast Anomaly Detection for Large Data Centers. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6, December 2010. ISSN: 1930-529X.
- [39] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, 2008.
- [40] Pierre Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49. JMLR Workshop and Conference Proceedings, June 2012. ISSN: 1938-7228.
- [41] Danilo Piparo, Enric Tejedor, Pere Mato, Luca Mascetti, Jakub Moscicki, and Mas-

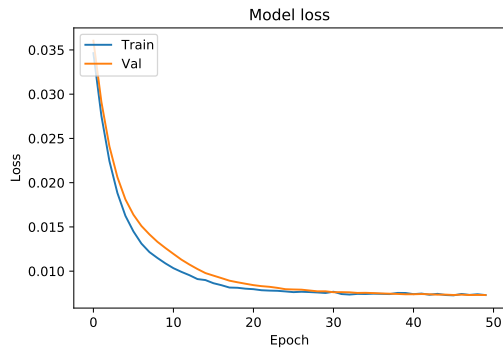
- simo Lamanna. Swan: A service for interactive analysis in the cloud. *Future Generation Computer Systems*, 78:1071–1078, 2018.
- [42] Meihui Jiang, Xiangyun Gao, Haizhong An, Huajiao Li, and Bowen Sun. Reconstructing complex network for characterizing the time-varying causality evolution behavior of multivariate time series. *Scientific Reports*, 7(1):10486, Sep 2017.
- [43] Richard E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 2015.
- [44] Rafael P. Álvarez, Carlos Giraldo, and David Chaves Dieguez. *Large scale anomaly detection in data center logs and metrics*. September 2018. Pages: 4.
- [45] Risto Vaarandi, Markus Kont, and Mauno Pihelgas. Event log analysis with the Log-Cluster tool. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 982–987, November 2016. ISSN: 2155-7586.
- [46] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, pages 1285–1298, New York, NY, USA, October 2017. Association for Computing Machinery.
- [47] Amrit Pal and Manish Kumar. DLME: Distributed Log Mining Using Ensemble Learning for Fault Prediction. *IEEE Systems Journal*, 13(4):3639–3650, December 2019. Conference Name: IEEE Systems Journal.
- [48] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1196–1201, November 2020. ISSN: 2374-8486.

List of Symbols

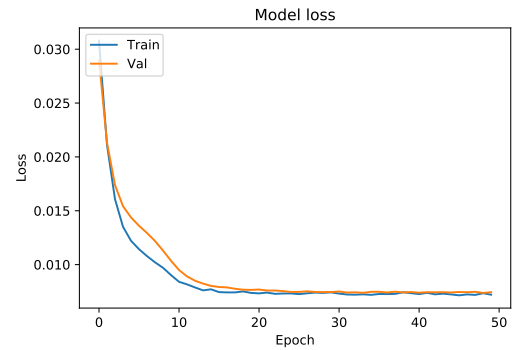
Symbol	Description
$\mathcal{C}$	The cluster containing similar servers
N	The total number of servers under study
M	The total number of metrics under study
$\mathcal{S}_i$	The i -th server of our cluster
$\mathbf{s}_i^j$	The time series related to the j -ith metric of a server $\mathcal{S}_i(t_k)$ at the time t_k
$\mathcal{P}_N$	The normal process generating normal time series
$\mathcal{P}_A$	The unknown process generating anomalous time series
τ	The time where the anomalous behavior starts
TR	The training set
$\mathbf{d}_i^j(t_k)$	The downsampled time series related to the j -ith metric of $\mathcal{S}_i$ at the time t_k
Δ	The resolution of the downsampling
$\mu_j(\mathcal{C})$	The mean value of the j -th metric w.r.t. the cluster $\mathcal{C}$
$\sigma_j(\mathcal{C})$	The standard deviation of the j -th metric w.r.t. the cluster $\mathcal{C}$
$\mathbf{n}_i^j(t_k)$	The final time series related to the j -ith metric of a server $\mathcal{S}_i$ at the time t_k
$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
L	The length of the time window considered for the AD task
$\mathbf{W}_{i,k}$	The window of normalized data between t_k and t_{k+L-1} of a server $\mathcal{S}_i$
$\mathcal{A}(\cdot)$	The anomaly score function
$AD(\mathbf{W}_{i,k}, \Gamma)$	The anomaly detection function
Γ	The threshold used for detecting anomalies
λ	The index related to the adopted models ($\lambda = 1, \dots, 3$)
$ENS_e(\mathbf{W}_{i,k})$	The e -th ensemble following the related voting strategy ($e = 1, \dots, 3$)

A | Appendix - Detailed Results

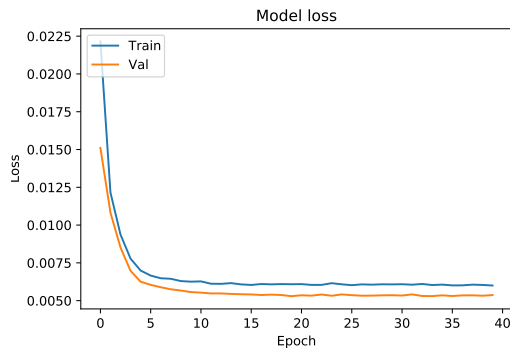
We include here those images that are too large to be included in the text.



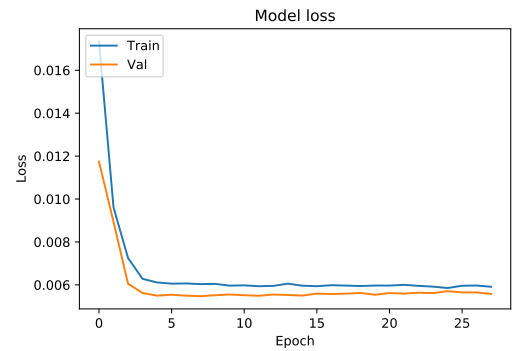
(a) LSTM, 1 week of training data.



(b) GRU, 1 week of training data.



(c) LSTM, 1 month of training data.



(d) GRU, 1 month of training data.

Figure A.1: The loss function of the autoencoders with different amount of training data.

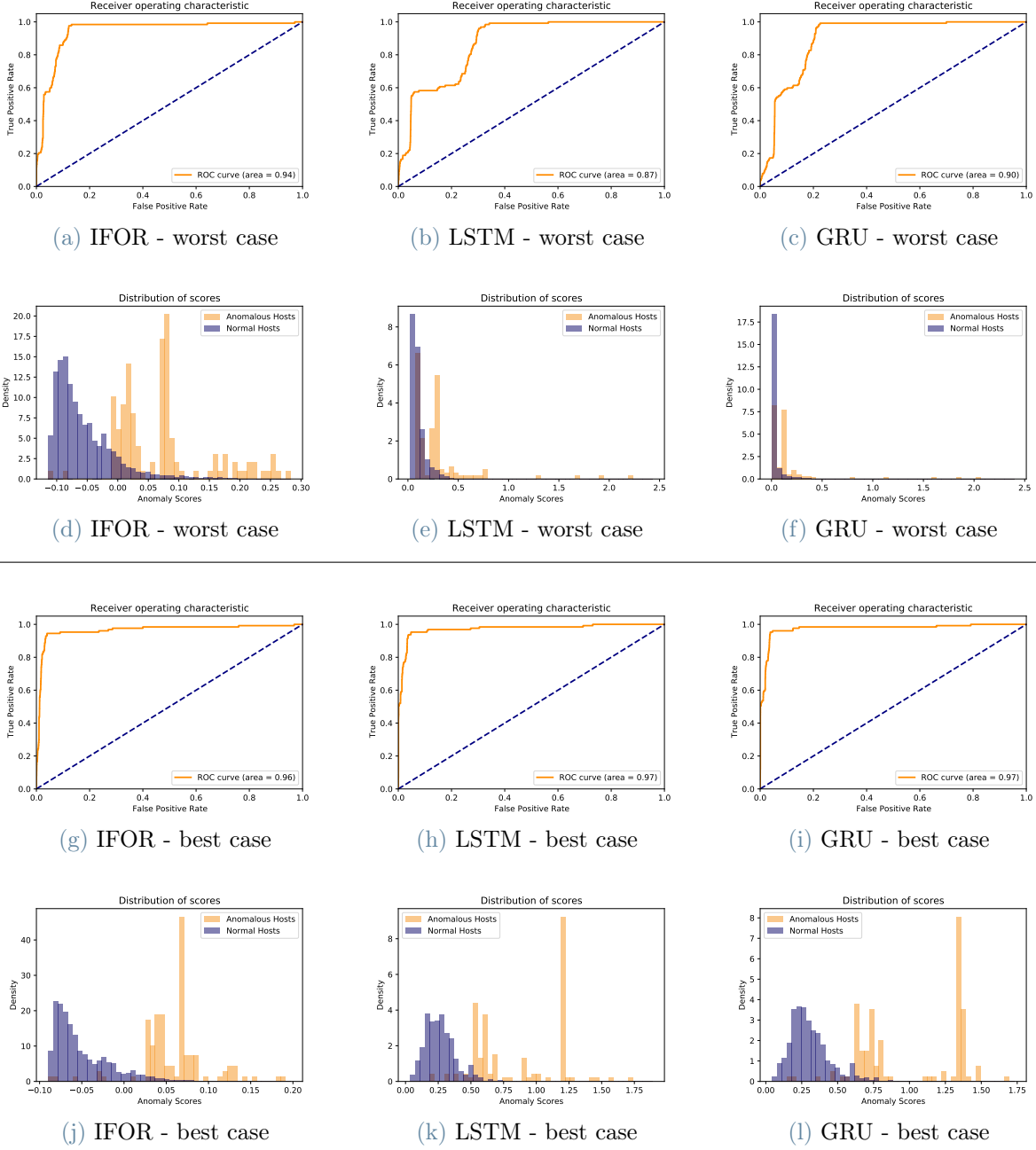


Figure A.2: The ROC curves and the score distributions of the three models that we consider. Notice that, for the worst cases, the models are trained with only one week of data and the parameters are not decided by the empirical results. On the opposite, the best cases are related to one month of training data and more robust parameters.



Figure A.3: A screenshot showing the old system dashboard. We can notice the different red thresholds for the metrics. If a metric is above its threshold the related server is identified as anomalous.

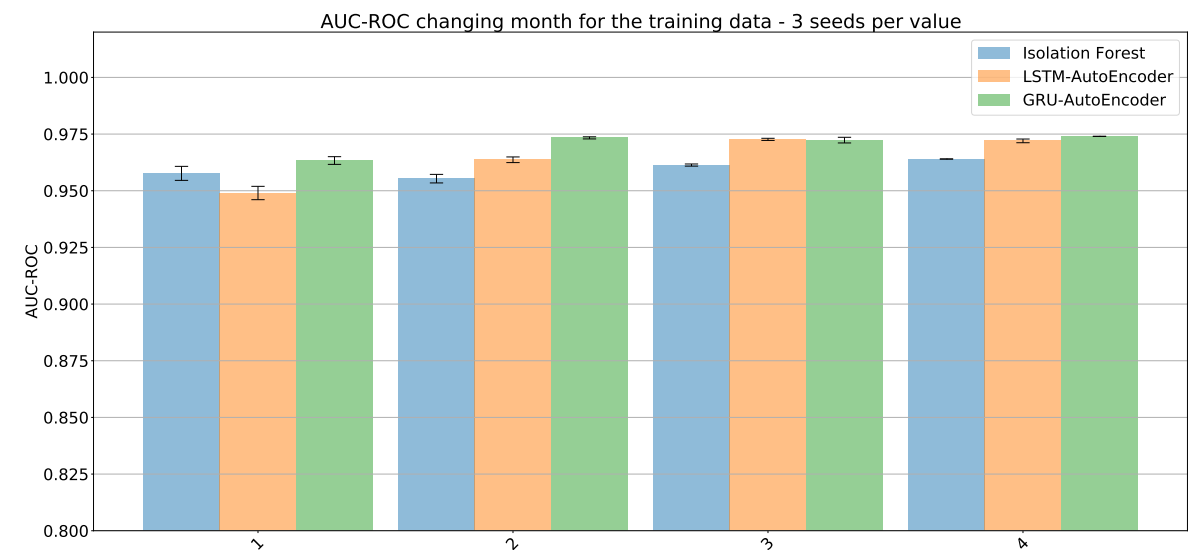


Figure A.4: The AUC-ROC of the three models that we consider changing the training set selection. Notice that here we are using one month of data for the training and we are shifting month for the next training set selection.

B | Appendix - Code Snippets

B.1. Utils

```

1 def read_local_window_dataset(config_filepath=None, path=None, nr_timeseries=None,
  ↳ list_metrics_to_keep=None):
2     # Read the window dataset from the local cache folder.
3     if (config_filepath is None) and (path is not None) and (nr_timeseries is not None):
4         local_path = path
5         nr_timeseries = nr_timeseries
6     else:
7         config_dict = read_json_config(config_filepath)
8         local_path = config_dict["local_cache_folder"] + "/" + config_dict["code_project_name"]
9         nr_timeseries = len(config_dict["selected_plugins"])
10    pdf = pd.read_parquet(local_path)
11    for c in pdf.columns:
12        if '_h' in c:
13            pdf[c] = pdf[c].apply(lambda x: float( '%.7f'%(x) ))
14    pdf = pdf.sort_values(by=["timestamp", "hostname"], ignore_index=True)
15    pdf.drop_duplicates(subset=["timestamp", "hostname"], inplace=True)
16    regex_history = re.compile("_h[0-9]*\$")
17    h_columns = list(filter(regex_history.search, pdf.columns))
18    feature_dataset = pdf[h_columns]
19    if list_metrics_to_keep is not None:
20        columns_to_keep = []
21        for metric in list_metrics_to_keep:
22            regex_this_metric = re.compile(metric + "_h[0-9]*\$")
23            columns_this_metric = list(filter(regex_this_metric.search, pdf.columns))
24            columns_to_keep += columns_this_metric
25        feature_dataset = feature_dataset[columns_to_keep]
26    provenance_dataset = pdf[["hostname", "ts", "timestamp"]]
27    feature_dataset.head()
28    provenance_dataset.head()
29    return feature_dataset, provenance_dataset, nr_timeseries

```

```

1 def read_json_config(local_path):
2     # Read the configuration file.
3     with open(local_path) as json_file:
4         config_dict = json.load(json_file)
5     return config_dict

```

```
def keep_only_existing_paths(spark, paths, local=False):  
    # Filter paths and discard the non-existent.  
  
    if not local:  
        sc = spark.sparkContext  
        hadoop = sc._jvm.org.apache.hadoop  
        fs = hadoop.fs.FileSystem  
        conf = hadoop.conf.Configuration()  
  
        existing_paths = []  
        non_existing = []  
  
        for path_str in paths:  
            if not local:  
                path = hadoop.fs.Path(path_str)  
                try:  
                    fs.get(conf).listStatus(path)  
                    existing_paths.append(path_str)  
                except Exception:  
                    non_existing.append(path_str)  
            else:  
                if os.path.isdir(path_str):  
                    existing_paths.append(path_str)  
                else:  
                    non_existing.append(path_str)  
  
        if len(existing_paths) == 0:  
            if not local:  
                raise Exception("In this configuration file you inserted only data that do not exist in  
                                ↳ HDFS")  
            else:  
                raise Exception("In this configuration file you inserted only data that do not exist in  
                                ↳ EOS")  
  
        return existing_paths
```

```

1 def filtered_plugins_hostgroups_df(df, plugins, hostgroups):
2     # Keep only the plugins and hostgroups.
3     only_my_hgs = df.filter(F.col("submitter_hostgroup").rlike(hostgroups[0]))
4     filter_str = ""
5     or_conditions = []
6     for k in plugins.keys():
7         plugin_dict = plugins[k]
8         plg_filter = "( " + plugin_dict['plugin_filter'] + " )"
9         or_conditions.append(plg_filter)
10    filter_str = " or ".join(or_conditions)
11    only_my_plugins = only_my_hgs.filter(filter_str)
12    df_list_plugin_ranamed = []
13    for plg_name, condition in zip(plugins.keys(), or_conditions):
14        df_current = only_my_plugins.filter(condition)
15        df_current = df_current.withColumn('plugin', F.lit(plg_name))
16        if "function" in plugins[plg_name]:
17            function_name = plugins[plg_name]['function']
18            df_current = df_current.withColumn("value", getattr(F, function_name)(col("value")))
19            df_current = reduce(lambda df_current, x: df_current.withColumn(x, F.when(F.col(x) < 0,
20                ↪ 0).otherwise(F.col(x))), ['value'], df_current)
21        df_list_plugin_ranamed.append(df_current)
22    only_my_plugins_renamed = union_all(df_list_plugin_ranamed)
23    relevant_cols = only_my_plugins_renamed.select("event_timestamp", "plugin", "host",
24        ↪ "submitter_hostgroup", "value").withColumnRenamed("event_timestamp",
25        ↪ "timestamp").withColumnRenamed("host", "hostname").withColumnRenamed("submitter_hostgroup",
26        ↪ "hostgroup").withColumn("timestamp", (col("timestamp") / 1000).cast("int"))
27    return relevant_cols

```

```

1 def get_date_between(start_date: str, end_date: str):
2     """Get date in the following range as tuples of int (year, month, day).
3
4     NB: the end_date is excluded
5     """
6     list_of_dates = []
7     sdate = datetime.strptime(start_date, "%Y-%m-%d") # start date
8     edate = datetime.strptime(end_date, "%Y-%m-%d") # end date
9     delta = edate - sdate # as timedelta
10    for i in range(delta.days):
11        day = sdate + timedelta(days=i)
12        day_tuple = (day.year, day.month, day.day)
13        list_of_dates.append(day_tuple)
14    return list_of_dates

```

B.2. ETL Steps

```

1 def data_reading(spark, plugins, days, hostgroups, local=False, schema_path=None):
2     # Read all plugin and all days.
3     plugins = copy.deepcopy(plugins)
4     inpaths = create_paths_for_plugins(plugins=plugins, days=days)
5     existing_inpaths = keep_only_existing_paths(spark=spark, paths=inpaths, local=local)
6     try:
7         if local:
8             existing_inpaths = ["file://" + x for x in inpaths]
9         if schema_path is not None:
10            schema = load_schema(schema_path)
11            df_raw = spark.read.schema(schema).parquet(*existing_inpaths)
12        else:
13            df_raw = spark.read.parquet(*existing_inpaths)
14    except AnalysisException:
15        return
16    all_data = filtered_plugins_hostgroups_df(df_raw, plugins, hostgroups)
17    return all_data

```

```

1 def downsampling_and_aggregate(df_all_plugins, every_n_minutes):
2     # Create the aggregate every x minutes.
3     aggregated_data = df_all_plugins.withColumn('minutes_group', F.col("timestamp") -
4     ↪ (F.col("timestamp") \% (60 * every_n_minutes))).withColumn('timestamp',
5     ↪ F.col("minutes_group") + (60 * every_n_minutes)).groupBy(['hostgroup', 'hostname', 'plugin',
6     ↪ 'minutes_group']).agg(F.mean('value').alias('value'),
7     ↪ F.max("timestamp").alias("timestamp")).orderBy("timestamp")
8     aggregated_data.show(5, False)
9     return aggregated_data

```

```

1 def normalize(df, df_normalization):
2     # Normalize the data given a dataframe with the coefficients.
3     df = df.persist()
4     df_normalization = df_normalization.persist()
5     to_be_normalized = df.join(df_normalization, 'plugin')
6     normalized_data = to_be_normalized.withColumn("value", (F.col("value") - F.col("mean")) /
7     ↪ F.col("stddev")).select("timestamp", "hostname", "hostgroup", "plugin", "value")
8     normalized_data.show(5, False)
9     return normalized_data

```

```

1 def pivot_plugin_as_column(df, plugin_names_list):
2     # Pivot your dataframe and get the plugins as column.
3     plugins_as_col = df.groupBy(["timestamp", "hostname", "hostgroup"]).pivot("plugin",
4     ↪ values=plugin_names_list).min("value")
5     plugins_as_col.show(5, False)
6     nonNull_cols = [c for c in plugins_as_col.columns if
7     ↪ plugins_as_col.filter(col(c).isNotNull()).count() > 0]
8     plugins_as_col = plugins_as_col.select(*nonNull_cols)
9     plugins_as_col.show(5, False)
10    return plugins_as_col

```



```

1 def compute_the_coefficients(df_aggregated):
2     # Compute from each of the plugin columns.
3     df_normalization = df_aggregated.groupBy("plugin").agg(F.avg('value').alias("mean"),
4     ↪ F.stddev('value').alias("stddev"))
5     df_normalization.show(5, False)
6     return df_normalization

```

```

1 def save_normalization(df_normalization, id_normalization,
2     outfolder, mode="overwrite", local=False):
3     # Save the normalization database in the folder.
4     norm_path = outfolder + "/" + id_normalization
5     if local:
6         norm_path = "file://" + norm_path
7     try:
8         df_normalization.write.format("parquet").mode(mode).save(norm_path)
9     except Exception as e:
10        return 1, None
11    return 0, norm_path

```

```

1 def create_window(df_plugins_as_col, steps):
2     # Create a window with the timesteps for history and past.
3     plugins = df_plugins_as_col.columns
4     [plugins.remove(x) for x in ['timestamp', 'hostname', 'hostgroup']]
5     window_spec = Window.partitionBy("hostname").orderBy("timestamp")
6     metrics_dfs = []
7     for metric in sorted(plugins):
8         df_metric = df_plugins_as_col
9         for lag_step in range(steps):
10            lag_col_name = metric + "_h" + str(lag_step)
11            df_metric = df_metric.withColumn(lag_col_name, F.lag(metric,
12            ↪ -lag_step).over(window_spec))
13            columns_to_clone = [x for x in df_metric.columns if x not in plugins]
14            metrics_dfs.append(df_metric.select(columns_to_clone))
15        df_window = join_all(dfs=metrics_dfs, attributes=["timestamp", "hostname", "hostgroup"])
16    return df_window

```

```

1 def data_writing(df_window, outfolder, mode='overwrite'):
2     # Save the windows partitioned in the outfolder.
3     try:
4         df_window.repartition(10).write.format("parquet").mode(mode).save(outfolder)
5     except Exception as e:
6         return 1
7     return 0, df_window

```

B.3. ETL Pipelines

```

1 def get_aggregated_data(spark, config_filepath, normalization=False, local=False):
2     config_dict = read_json_config(config_filepath)
3     plugins = config_dict["selected_plugins"]
4     if normalization:
5         days = get_date_between(config_dict["date_start_normalization"],
6                                 ↪ config_dict["date_end_normalization_excluded"])
7     else:
8         days = get_date_between(config_dict["date_start"], config_dict["date_end_excluded"])
9     hostgroups = config_dict["hostgroups"]
10    try:
11        schema_path = config_dict["schemas"]["raw_path"]
12    except Exception as e:
13        schema_path = None
14    all_raw_data = data_reading(spark=spark, plugins=plugins, days=days, hostgroups=hostgroups,
15                                ↪ local=local, schema_path=schema_path)
16    every_n_minutes = config_dict["aggregate_every_n_minutes"]
17    agg_data = downsampling_and_aggregate(df_all_plugins=all_raw_data,
18                                          ↪ every_n_minutes=every_n_minutes)
19    return agg_data

```

```

1 def pipeline_preparation_norm_coeff(spark, config_filepath, local=False):
2     # Run the pipeline to get the coefficient and create a normalization df.
3     config_dict = read_json_config(config_filepath)
4     plugins = config_dict["selected_plugins"]
5     hostgroups = config_dict["hostgroups"]
6     agg_data = get_aggregated_data(spark=spark, config_filepath=config_filepath, normalization=True,
7                                     ↪ local=local)
8     normalized_data = compute_the_coefficients(df_aggregated=agg_data)
9     id_norm = create_id_for_hostgroups_and_plugins_start_end(hostgroups=hostgroups, plugins=plugins,
10                                                              ↪ start=config_dict["date_start_normalization"],
11                                                              ↪ end=config_dict["date_end_normalization_excluded"])
12     outfolder = config_dict["normalization_out_folder"]
13    try:
14        if config_dict["overwrite_on_hdfs"] is True:
15            mode = "overwrite"
16        elif config_dict["overwrite_on_hdfs"] is False:
17            mode = "append"
18    except Exception as e:
19        mode = "overwrite"
20    exit_code, norm_path = save_normalization(df_normalization=normalized_data,
21                                              ↪ id_normalization=id_norm, outfolder=outfolder, mode=mode, local=local)
22    return exit_code, norm_path

```

```

1 def read_normalization_dataframe(spark, normalization_folder: str, normalization_id: str,
  ↪ schema_path=None):
2     # Get the normalization dataframe with the corresponding id.
3     df_normalization = read_spark_df(spark=spark, df_path=normalization_folder + "/" +
  ↪ normalization_id, schema_path=schema_path)
4     return df_normalization

```

```

1 def run_pipeline_all_in_one(spark, config_filepath, local=False):
2     # Run the pipeline for the specified data.
3     config_dict = read_json_config(config_filepath)
4     plugins = config_dict["selected_plugins"]
5     hostgroups = config_dict["hostgroups"]
6     normalization_id = create_id_for_hostgroups_and_plugins_start_end(hostgroups=hostgroups,
  ↪ plugins=plugins, start=config_dict["date_start_normalization"],
  ↪ end=config_dict["date_end_normalization_excluded"])
7     try:
8         schema_path = config_dict["schemas"]["norm_path"]
9     except Exception as e:
10         schema_path = None
11     norm_df = read_normalization_dataframe(spark=spark,
  ↪ normalization_folder=config_dict["normalization_out_folder"],
  ↪ normalization_id=normalization_id, schema_path=schema_path)
12     agg_data = get_aggregated_data(spark=spark, config_filepath=config_filepath, normalization=False,
  ↪ local=local)
13     normalized_data = normalize(df=agg_data, df_normalization=norm_df)
14     pivotted_data = pivot_plugin_as_column(df=normalized_data,
  ↪ plugin_names_list=list(plugins.keys()))
15     history_steps = config_dict["history_steps"]
16     windows_data = create_window(df_plugins_as_col=pivotted_data, steps=history_steps)
17     outfolder = config_dict["hdfs_out_folder"] + "/" + config_dict["code_project_name"]
18     try:
19         if config_dict["overwrite_on_hdfs"] is True:
20             mode = "overwrite"
21         elif config_dict["overwrite_on_hdfs"] is False:
22             mode = "append"
23     except Exception as e:
24         mode = "overwrite"
25     if not local:
26         try:
27             subprocess.call(["hdfs", "dfs", "-rm", "-R", "-skipTrash", outfolder])
28         except Exception as e_delete:
29             return
30     exit_code, final_df = data_writing(df_window=windows_data, outfolder=outfolder, mode=mode)
31     return exit_code, final_df

```

```

1 def materialize_locally(spark, config_filepath, local=False):
2     # Create a window dataset and move it locally."""
3     config_dict = read_json_config(config_filepath)
4     project_code = config_dict["code_project_name"]
5     hdfs_outfolder = config_dict["hdfs_cache_folder"] + "/" + project_code
6     local_outfolder = config_dict["local_cache_folder"] + "/" + project_code
7     df_win_train = read_window_dataset(spark=spark, config_filepath=config_filepath)
8     if local:
9         hdfs_outfolder = "file://" + hdfs_outfolder
10    df_win_train.coalesce(1).write.format("parquet").mode("overwrite").save(hdfs_outfolder)
11    if not local:
12        copy_to_local(hdfs_path=hdfs_outfolder, local_path=local_outfolder)
13        raw_folder = config_dict["hdfs_out_folder"] + project_code
14        try:
15            subprocess.call(["hdfs", "dfs", "-rm", "-R", "-skipTrash", raw_folder])
16        except Exception as e_delete:
17            return
18        try:
19            subprocess.call(["hdfs", "dfs", "-rm", "-R", "-skipTrash", hdfs_outfolder])
20        except Exception as e_delete:
21            return

```

```

1 def read_spark_df(spark, df_path, schema_path=None):
2     # Reading a spark dataframe with the possibility of using the schema.
3     if schema_path is not None:
4         schema = load_schema(schema_path)
5         try:
6             df = spark.read.schema(schema).parquet(df_path)
7         except Exception as e:
8             raise
9     else:
10        try:
11            df = spark.read.parquet(df_path)
12        except Exception as e:
13            raise
14    try:
15        df.show(5)
16    except Exception as e:
17        raise
18    return df

```

```

1 def get_normalization_pandas(spark, config_filepath, local=False, schema_path=None):
2     # Return the Pandas dataframe of normalization.
3     path = get_normalization_path(config_filepath=config_filepath)
4     if local:
5         path = "file://" + path
6     df = read_spark_df(spark=spark, df_path=path, schema_path=schema_path)
7     pdf = df.toPandas()
8     return pdf

```

B.4. PyOD Wrapper

```

1 class BaseOutlierAnalyser(ABC):
2     @abstractmethod
3     def __init__(self):
4         # Abstract class for all outlier detection algorithms.
5
6     @abstractmethod
7     def fit(self, X, prov, y=None, value_for_missing=0):
8         # Fit the model using X as training data.
9         pass
10
11     @abstractmethod
12     def predict(self, X, prov, y=None, value_for_missing=0):
13         # Predict the scores for the X testing data.
14         pass
15
16     def get_scores(self, on_machines=None):
17         # Get the score of the machines analyzed.
18         return self.scores
19
20     def get_machines(self):
21         # Get the names of the machines analyzed.
22         return self.host_names
23
24     def _compute_ranking(self):
25         # Compute the ranking of the scores.
26         ordered_index = np.argsort(self.scores)[::-1]
27         self.ordered_scores = [self.scores[i] for i in ordered_index]
28         self.ordered_host_names = [self.host_names[i] for i in ordered_index]
29         self.ranks = [sorted(self.scores, reverse=True).index(x) + 1 for x in self.scores]
30         self.percentiles = [scipy.stats.percentileofscore(self.scores, a, 'weak') for a in
31                               ↪ self.scores]
32
33     def prepare_to_publish_config(self, config_file, slide_steps, algo_name):
34         # Give all the info necessary to publish.
35         config_dictionary = read_json_config(config_file)
36         hostgroup = config_dictionary["hostgroups"][0]
37         self.cell_name = hostgroup[hostgroup.rfind("/") + 1:]
38         self.hostgroup = hostgroup
39         self.granularity_min = config_dictionary["aggregate_every_n_minutes"]
40         self.plugins = config_dictionary["selected_plugins"]
41         self.normalization_name = "standardization"
42         self.normalization_id =
43             ↪ create_id_for_hostgroups_and_plugins_start_end(hostgroups=config_dictionary["hostgroups"],
44             ↪ plugins=config_dictionary["selected_plugins"],
45             ↪ start=config_dictionary["date_start_normalization"],
46             ↪ end=config_dictionary["date_end_normalization_excluded"])
47         self.slide_steps = slide_steps
48         self.history_len = config_dictionary["history_steps"]
49         self.classname = self.__class__.__name__
50         self.algo_name = algo_name

```

```

1 class PyODWrapperAnalyzer(BaseOutlierAnalyser):
2     def __init__(self, pyod_detector, normalization_anomaly_scores=True, robust=False):
3         # Create the wrapper of a PyOD detector.
4         self.pyod_detector = pyod_detector
5         self.robust = robust
6         self.normalization_anomaly_scores = normalization_anomaly_scores
7
8     def fit(self, X, prov, y=None, value_for_missing=0, forward_prov=False):
9         # Fit the model using X as training data.
10        X_cleaned = X.fillna(value_for_missing)
11        if forward_prov:
12            self.pyod_detector.fit(X_cleaned, prov=prov)
13        else:
14            self.pyod_detector.fit(X_cleaned)
15        if self.normalization_anomaly_scores:
16            self._compute_coefficients_std_score()
17
18    def predict(self, X, prov, y=None, value_for_missing=0):
19        # Predict the model using X as testig data.
20        X_cleaned = X.fillna(value_for_missing)
21        self.scores = self.pyod_detector.decision_function(X_cleaned)
22        self.std_scores = self.scores
23        if len(prov.shape) == 2:
24            self.host_names = [h for h in prov.iloc[:, 0].values.flatten()]
25        else:
26            self.host_names = prov[0]
27
28    def get_std_train_scores(self):
29        # Get all the scores standardized during training.
30        train_scores = self.pyod_detector.decision_scores_
31        return self.standardize_with_train(train_scores)
32
33    def get_train_scores(self):
34        # Get the anomaly scores on the train set.
35        return self.pyod_detector.decision_scores_
36
37    def predict_std(self, X, prov, y=None, value_for_missing=0):
38        # Predict the scores and standardize with training scores.
39        self.predict(X, prov, y, value_for_missing)
40        std_scores = self.standardize_with_train(self.scores)
41        self.std_scores = std_scores

```

B.5. GRU-AutoEncoder

```

1 class AEGruTF2(BaseDetector):
2     def __init__(self, nr_timesteps, patience=None, loss=mean_squared_error, optimizer='adam',
    ↪ gru_units=1, epochs=10, batch_size=16, dropout_rate=0.25, validation_size=0.1, verbose=1,
    ↪ random_state=None, contamination=0.1, preprocessing=True):
3         ...
4
5     def _build_model(self, gru_units):
6         if self.random_state is not None:
7             fix_seeds(self.random_state)
8         model = Sequential()
9         model.add(GRU(gru_units, return_sequences=True, input_shape=(self.nr_timesteps,
    ↪ self.nr_timeseries)))
10        model.add(Dropout(self.dropout_rate))
11        model.add(GRU(gru_units, return_sequences=True))
12        model.add(Dropout(self.dropout_rate))
13        model.add(TimeDistributed(Dense(self.nr_timeseries)))
14        model.compile(optimizer=self.optimizer, loss=self.loss)
15        return model
16
17    def fit(self, X, y=None):
18        self.plugin_names = list(set([x[:x.rfind("_")] for x in X.columns]))
19        self.nr_timeseries = len(self.plugin_names)
20        X_image = np.reshape(X.values, (len(X), self.nr_timeseries, self.nr_timesteps))
21        X_image = np.nan_to_num(X_image, nan=0, posinf=0, neginf=0)
22        if self.preprocessing:
23            self.v_min = X_image.min(axis=(0, 2), keepdims=True)
24            self.v_max = X_image.max(axis=(0, 2), keepdims=True)
25            denominator = (self.v_max - self.v_min)
26            denominator[denominator == 0] = 1
27            X_image = (X_image - self.v_min) / denominator
28        X_image = X_image.swapaxes(1, 2)
29        self.model_ = self._build_model(gru_units=self.gru_units)
30        early_stop = EarlyStopping(monitor = 'val_loss', patience = self.patience)
31        self.history_ = self.model_.fit(X_image, X_image, epochs=self.epochs,
    ↪ batch_size=self.batch_size, shuffle=True, validation_split=self.validation_size,
    ↪ verbose=self.verbose, callbacks = [early_stop]).history
32        pred_scores = self.model_.predict(X_image)
33        nr_rows = pred_scores.shape[0]
34        pred_scores_flat = np.reshape(pred_scores, (nr_rows, self.nr_timeseries * self.nr_timesteps))
35        original_images = np.reshape(X_image, (nr_rows, self.nr_timeseries * self.nr_timesteps))
36        self.decision_scores_ = pairwise_distances_no_broadcast(original_images, pred_scores_flat)
37        self._process_decision_scores()
38        return self
39
40    def decision_function(self, X):
41        check_is_fitted(self, ['model_', 'history_'])
42        X_image = np.reshape(X.values, (len(X), self.nr_timeseries, self.nr_timesteps))
43        X_image = np.nan_to_num(X_image, nan=0, posinf=0, neginf=0)
44        if self.preprocessing:
45            denominator = (self.v_max - self.v_min)
46            denominator[denominator == 0] = 1
47        ...

```

B.6. Airflow

```

1 def ad_dag(dag_id='give_me_a_name', override_params={}):
2     default_params={
3         'default_args': {
4             'owner': 'Airflow',
5             'start_date': datetime(2021, 1, 1),
6             'ad_config_train': {...},
7             'ad_config_inference': {...},
8             'ad_config_experiment': {...},
9         },
10        'schedule_interval': '0 0 1 * *',
11        'max_active_runs': 1, # number active runs
12        'concurrency': 1, # total tasks concurrency
13        'dagrun_timeout': timedelta(minutes=120),
14        'tags': ['Airflow_Production'],
15    }
16    dag_args = merge({'dag_id':dag_id}, default_params, override_params)
17    dag = DAG(**dag_args)
18    dag_exit_status = PythonOperator(task_id='dag_exit_status', provide_context=True,
19    ↪ python_callable=final_status, trigger_rule='all_done', dag=dag)
20
21    ...
22
23    tg_train = TaskGroup(group_id='TRN', dag=dag)
24    tg_train_models = TaskGroup(group_id='TRN_', dag=dag)
25    tg_inference = TaskGroup(group_id='INF', dag=dag)
26    tg_inference_scores = TaskGroup(group_id='INF_', dag=dag)
27    tg_train >> tg_train_models >> tg_inference >> tg_inference_scores
28    for atask in ad_tasks_train:
29        globals()[atask[0]] = return_configured_BashOperator(*atask, task_group=tg_train)
30    for atask in ad_task_train_models:
31        globals()[atask[0]] = return_configured_BashOperator(*atask, task_group=tg_train_models)
32    for atask in ad_tasks_inference:
33        globals()[atask[0]] = return_configured_BashOperator(*atask, task_group=tg_inference)
34    for atask in ad_task_inference_scores:
35        globals()[atask[0]] = return_configured_BashOperator(*atask, task_group=tg_inference_scores)
36
37    check_local_data_presence_TRN >> train_models
38    check_local_data_presence_TRN >> spark_check_norm_presence_TRN
39    spark_check_norm_presence_TRN >> spark_compute_norm_TRN >> spark_transform_data_TRN
40    spark_check_norm_presence_TRN >> spark_transform_data_TRN
41    spark_transform_data_TRN >> spark_mv_data_to_local_TRN
42    spark_mv_data_to_local_TRN >> train_models
43    train_models >> check_local_data_presence_INF
44    check_local_data_presence_INF >> inference_scores
45    check_local_data_presence_INF >> spark_check_norm_presence_INF
46    spark_check_norm_presence_INF >> spark_compute_norm_INF >> spark_transform_data_INF
47    spark_check_norm_presence_INF >> spark_transform_data_INF
48    spark_transform_data_INF >> spark_mv_data_to_local_INF
49    spark_mv_data_to_local_INF >> inference_scores
50    inference_scores >> dag_exit_status
51    return dag

```


B.7. Unit Tests

```

1 class TestEtlSteps(unittest.TestCase):
2     def setUp(self):
3         os.environ['PYSPARK_PYTHON'] = sys.executable
4         os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
5         number_cores = 1
6         memory_gb = 3
7         conf =
8             ↪ (pyspark.SparkConf().setMaster('local[{}]'.format(number_cores)).set('spark.driver.memory',
9             ↪ '{}g'.format(memory_gb)))
10        self.sc = pyspark.SparkContext(conf=conf)
11        self.spark = pyspark.sql.SparkSession(self.sc)
12        test_file = os.path.dirname(os.path.realpath(__file__)) + "/test.yaml"
13        with open(test_file) as yaml_file:
14            self.file_dict = yaml.safe_load(yaml_file)
15
16    def test__filtered_plugins_hostgroups_df__compare(self):
17        df = self.spark.read.parquet("file://" + self.file_dict['df_raw_path'])
18        df_to_compare = filtered_plugins_hostgroups_df(df=df,
19            ↪ plugins=self.file_dict['selected_plugins'],
20            ↪ hostgroups=self.file_dict['filtered_plugins_hostgroups_df_HOSTGROUPS_TO_FILTER'])
21        outfolder = self.file_dict['df_filtered_path']
22        expectation = self.spark.read.parquet("file://" + outfolder)
23        self.assertTrue(are_dfs_equal(df_to_compare, expectation))
24
25    def test__PLUGIN_FUNCTIONS_filtered_plugins_hostgroups_df__compare(self):
26        df = self.spark.read.parquet("file://" + self.file_dict['df_raw_path'])
27        df_to_compare = filtered_plugins_hostgroups_df(df=df,
28            ↪ plugins=self.file_dict['PLUGIN_FUNCTIONS_selected_plugins'],
29            ↪ hostgroups=self.file_dict['filtered_plugins_hostgroups_df_HOSTGROUPS_TO_FILTER'])
30        outfolder = self.file_dict['df_filtered_path']
31        expectation = self.spark.read.parquet("file://" + outfolder)
32        self.assertTrue(are_dfs_equal(df_to_compare, expectation))
33
34    def test__data_reading__compare(self):
35        df_to_compare = data_reading(spark=self.spark, plugins=self.file_dict["selected_plugins"],
36            ↪ days=get_date_between(start_date=self.file_dict["start_date"],
37            ↪ end_date=self.file_dict["end_date"]),
38            ↪ hostgroups=self.file_dict["data_reading_HOSTGROUPS_TO_FILTER"], local=True,
39            ↪ schema_path=self.file_dict['json_load_schema_paths']['raw'])
40        outfolder = self.file_dict['df_filtered_path']
41        expectation = self.spark.read.parquet("file://" + outfolder)
42        self.assertTrue(are_dfs_equal(df_to_compare, expectation))

```

