RUAD: unsupervised anomaly detection in HPC systems

Martin Molan[a], Andrea Borghesi[a], Daniele Cesarini[b], Luca Benini[a,c], Andrea Bartolini[a]

[a] *DISI and DEI Department, University of Bologna, Bologna, Italy*
[b] *CINECA consorzio interuniversitario, Bologna, Italy*
[c] *Institut für Integrierte Systeme, ETH, Zürich, Switzerland*

Abstract

The increasing complexity of modern high-performance computing (HPC) systems necessitates the introduction of automated and data-driven methodologies to support system administrators' effort toward increasing the system's availability. Anomaly detection is an integral part of improving the availability as it eases the system administrator's burden and reduces the time between an anomaly and its resolution. However, current state-of-the-art (SOTA) approaches to anomaly detection are supervised and semi-supervised, so they require a human-labelled dataset with anomalies - this is often impractical to collect in production HPC systems. Unsupervised anomaly detection approaches based on clustering, aimed at alleviating the need for accurate anomaly data, have so far shown poor performance.

In this work, we overcome these limitations by proposing RUAD, a novel Recurrent Unsupervised Anomaly Detection model. RUAD achieves better results than the current semi-supervised and unsupervised SOTA approaches. This is achieved by considering temporal dependencies in the data and including long-short term memory cells in the model architecture. The proposed approach is assessed on a complete ten-month history of a Tier-0 system (Marconi100 from CINECA with 980 nodes). RUAD achieves an area under the curve (AUC) of 0.763 in semi-supervised training and an AUC of 0.767 in unsupervised training, which improves upon the SOTA approach that achieves an AUC of 0.747 in semi-supervised training and an AUC of 0.734 in unsupervised training. It also vastly outperforms the current SOTA unsupervised anomaly detection approach based on clustering, achieving the AUC of 0.548.

1. Introduction

Recent trends in the development of high-performance computing (HPC) systems (such as heterogeneous architecture and higher-power integration density) have increased the complexity of their management and maintenance [1]. A typical contemporary HPC system consists of thousands of interconnected nodes; each node usually contains multiple different accelerators such as graphical processors, FPGAs, and tensor cores [2]. Monitoring the health of all those subsystems is an increasingly daunting task for system administrators. To simplify this monitoring task and reduce the time between anomaly insurgency and response by the administrators, automatic anomaly detection systems have been introduced in recent years [3].

Anomalies that result in downtime or unavailability of the system are expensive events. Their cost is primarily associated with the time when the HPC system cannot accept new compute jobs. Since HPC systems are costly and have a limited service lifespan [4], it is in the interest of the system's operator to reduce unavailability times. Anomaly detection helps in this regard as it can significantly reduce the time between the fault and the response by the system administrator, compared to manual reporting of faulty nodes [5].

Modern supercomputers are endowed with monitoring systems that give the system administrators a holistic view of the system [3]. Data collected by these monitoring systems and historical data describing system availability are the basis for Machine Learning anomaly detection approaches [6, 7, 8, 9, 10], which build data-driven models of the supercomputer and its computing nodes. In this work, we focus on CINECA Tier0 HPC system (Marconi100 [11, 12] ranked 9th in Jun. 2020 Top500 list [13]), which employs a holistic monitoring system called EXAMON [14].

Production HPC systems are reliable machines that generally have very few downtime events - for instance, in Marconi100 at CINECA, timestamps corresponding to faulty events represent, on average, only 0.035% of all data. However, although anomalies are rare events, they still significantly impact the system's overall availability - during the observation period, there was at least one active anomaly (unavailable node) 14.4% of the time. State-of-the-art (SOTA) methods for anomaly detection on HPC systems are based on supervised and semi-supervised approaches from the Deep Learning (DL) field [5]; for this reason, these methods require a training set with accurately annotated periods of downtime (or anomalies). In turn, this requires the monitoring infrastructure to track downtime events; in some instances, this can be done with specific software tools (e.g., Nagios [15]), but properly configuring these tools is a complex and time-consuming task for system administrators.

So far, the challenges of anomaly detection on HPC systems have been approached by deploying anomaly reporting tools by training the models in a supervised or semi-supervised fashion [5, 16, 17, 8]. The need for an accurately labelled training set is the main limitation of current approaches as it is expensive, in terms of time and effort of the system administrators, to be applied in practice. Downtime tracking also has to be able to record failures with the same granularity as the other monitoring services. Some methods in production HPC systems only record downtime events by date [1, 2, 3]. In most production HPC systems, accurate anomaly detection is thus not readily achievable. For this reason, the majority of the methods from the literature were tested on historical or synthetic data or in supercomputers where faults were injected in a carefully controlled fashion [18]. Another limitation for the curation of an accurately labeled anomaly dataset is the short lifetime of most HPC systems. In the HPC sector, a given computing node and system technology have a lifetime of between three and five years. Short lifetime means, in practice, that the vendor has no time to create a dataset for training an anomaly detection model before the system is deployed to the customer site.

A completely unsupervised anomaly detection approach could be deployed on a new node or even on an entirely new HPC system. It would then learn online and without any interaction with the system administrators. Additionally, such a system would be easier to deploy as it would require no additional framework to report and record anomalous events (in addition to the monitoring infrastructure needed to build the data-driven model of the target supercomputer - a type of infrastructure which is becoming more and more widespread in current HPC facilities [3]).

Unsupervised anomaly detection approaches for HPC systems exist such as [19, 20, 21]. They either work on log or sensor data. Approaches based on log data [19, 21], while useful, can only offer a post-mortem and restricted view of the supercomputer state. The SOTA for anomaly detection on sensor data [20] is based on clustering, which requires a degree of manual analysis from system administrators and offers poor performance compared to semi-supervised methods. The semi-supervised methods [5, 6, 22], based on the dense autoencoders, which are trained to reproduce their input, could be trained in an unsupervised fashion. However, none of the presented works has explored this possibility. According to the SOTA, the models would perform worse as the dense autoencoder is also capable of learning the characteristics of the anomalies [5, 6, 22].

The primary motivation for this work is to propose a novel approach that relies *only on the fact that the anomalies are rare events* and works at least equally well when trained in an *unsupervised manner* as it does when trained in *semi-supervised manner* - this has not been the case in the current SOTA. In this work, we propose an *unsupervised* approach: RUAD (Recurrent Unsupervised Anomaly Detection) that works on sensor data and *outperforms* all other approaches, including the current SOTA semi-supervised approach [5] and SOTA unsupervised approach [20]. RUAD achieves that by taking into account temporal dependencies in the data. We achieve that by using Long Short-Term Memory (LSTM) cells in the proposed neural network model structure, which explicitly take into consideration the temporal dimension of observed phenomena. We also show that the RUAD model, comprising of LSTM layers, is capable of learning the characteristic of the normal operation even if the anomalous data is present in the test set - the RUAD model is thus able to be trained in an *unsupervised manner*. RUAD targets single HPC computing nodes: we have different anomaly detection models for each computing node. The motivation behind this is *scalability*: in this way, each node can be used to train its own model with minimal overhead - moreover, this strategy would work in larger supercomputers as well, as if the number of nodes increases, we just have to add new detection models.

1.1. Contributions of the paper

To recap, in this paper, we propose an anomaly detection framework that can handle complex system monitoring data, scale to large-scale HPC systems, and be trained even if no labelled dataset is available. The key contributions presented in this paper are:

- We propose a completely *unsupervised* anomaly detection approach (RUAD) that exploits the fact that the anomalies are rare and explicitly considers the *temporal dependencies* in the data by using LSTM cells in an autoencoder network. The resulting Deep Learning model *outperforms* the previous state-of-the-art semi-supervised approach [5], based on time-unaware autoencoder networks. On the dataset presented and analysed in this paper (collected from the Marconi100 supercomputer), the previous approach achieves an Area-Under-the-Curve (ACU) test set score of 0.7470. In contrast, our unsupervised approach achieves the best test set AUC score of 0.7672. To the best of our knowledge, this work is the first time such an approach has been applied to the field of HPC system monitoring and anomaly detection.

- We have conducted a *very large-scale experimental evaluation* of our methods. We have trained four different deep learning models for each of the 980+ nodes of Marconi100. To the best of our knowledge, this is the largest scale experiment relating to anomaly detection in HPC systems, both in terms of the number of considered nodes and length of time. Previous works only evaluate the models on a subset of nodes with a short observation time ([5], for instance, only analyzed 20 nodes of the HPC system over two months). Per-node training of models also demonstrates the feasibility of *per node* models for large HPC systems. The training time for the individual model was under 30 minutes on a single NVIDIA Volta V100 GPU.

1.2. Structure of the paper

We present the current state-of-the-art and position our paper in Section 2. The machine learning approaches used for anomaly detection, including our novel approach, are described in section 3. The experimental setting for empirical validation of our results is detailed in Section 4.1 and our results are discussed in the rest of Section 4. Finally, Section 5 offers some concluding remarks.

2. Related Works

The drive to detect events or instances that deviate from the norm (i.e. *operational anomalies*) is present across many industrial applications. One of the earliest applications of anomaly detection models was credit card fraud detection in the financial industry [23, 24]. Recently, anomaly detection (and associated predictive maintenance) has become relevant in manufacturing industries [25, 26], internet of things (IoT) [27, 28, 29], energy sector [30], medical diagnostics [31, 32], IT security [33], and even in complex physics experiments [34].

Typically, anomalies in an HPC system refer to periods of (and leading to) suboptimal operating modes, faults that lead to failed or incorrectly completed jobs, or node and other components hardware failures. While HPC systems have several possible failure mitigation strategies [35] and fault tolerance strategies [36], anomalies of this type still significantly reduce the amount of compute time available to users [37]. The transition towards Exascale and the increasing heterogeneity of hardware components will only exacerbate the issues stemming from failures, and anomalous conditions that already plague HPC machines [1, 3, 38]. A DARPA study estimates that the failures in future exascale HPC systems could occur as frequently as once every 35-39 minutes [39], thus significantly impacting the supercomputing availability and system administrator load.

However, when looking at specific components and not at the entire HPC system (e.g., considering a single computing node), faults remain very rare events, thus falling under the area of anomaly detection, which can be seen as an extreme case of supervised learning on unbalanced classes [40]. Because data regarding normal operation far exceeds data regarding anomalies, classical supervised learning approaches tend to overfit the normal data and give a sub-optimal performance on the anomalous data [41]. In order to mitigate the problem of unbalanced classes, the anomaly detection problem is typically approached from two angles. Approaches found in the State-of-Art (SOTA) that address the class imbalance either modify the data [42] or use specialized techniques that work well on anomaly detection problems [5]. Data manipulation approaches address the dataset imbalance either by decreasing the data belonging to normal operation (*under sampling the majority class*) or by oversampling or even generating anomalous data (*over sampling minority class*) [42]. Data manipulation for anomaly detection in HPC systems has not yet been thoroughly studied. Conversely, most existing approaches rely on synthetic data generation, e.g., injection of anomalies in real (non-production) supercomputers or HPC simulators [5].

Another research avenue exploits the abundance of normal data from HPC systems using a different learning strategy, , namely semi-supervised ML models. Instead of learning on a dataset containing multiple classes – and consequently learning the characteristics of all classes – semi-supervised models are trained only on the normal data. Hence, they are trained to learn the characteristics of the of the normal class (the majority class in the dataset). Anomalies are then recognized as anything that does not correspond to the learned characteristic of the normal class [40, 6, 43, 22, 44].

Regarding the type of data used to develop and deploy anomaly detection systems, we can identify two macro-classes: system monitoring data collected by holistic monitoring systems (i.e. Examon [14]) and log data. This data is then annotated with information about the system or node-level availability, thus creating a label associated with the data points. The label encodes whether the system is operating normally or experiencing an anomaly. Since it is expensive and time-consuming to obtain labelled system monitoring data, a labelled dataset for supervised learning can be obtained by "injecting" anomalies into the HPC system (like [18]). Labels are important for both supervised, semi-supervised and unsupervised approaches. In the first case, they are used to compute the loss, in the second case to identify the training dataset and validation, and in the third case, only for validation. This data can then be used in a supervised learning task directly or after processing new features (feature construction). Examples of this approach are [45, 17, 46] where authors use supervised ML approaches to classify the performance variations and job-level faults in HPC systems. For fault detection, [8, 18] propose a supervised approach based on Random Forest (an ensemble method based on decision trees) to classify faults in an HPC system. All mentioned approaches use synthetic anomalies injected into the HPC system to train a supervised classification model. Approaches [5] and [16] are among the few that leverage *real* anomalies collected from production HPC systems (as opposed to injected anomalies). In this paper, we are interested in real anomalies, and thus, we will not include methods using synthetic/simulated data or injected anomalies in our quantitative comparisons.

All mentioned approaches do not take into account temporal dependencies of data (models are not trained on time series but on *tabular data containing no temporal information*). System monitoring data approach [47] is the first to take into account temporal dependencies in data by calculating statistical features on temporal dimension (aggregation, sliding window statistics, lag features). Most approaches that deal with *time series anomaly detection* do so on system log data. Labelled anomalies are either analyzed with log parsers [48] or detected with deep learning methods. Deep learning methods for anomaly detection are based on LSTM neural networks as they are a proven approach in other text processing fields.

Compared to labelled training sets, much less work has been done on unlabelled datasets - despite this case being much more common in practice. So far, all research on unlabelled datasets has focused on system log data. [19] propose a *k*-means based unsupervised learning approach that does not take into account temporal dynamics of the log data. A clustering-based approach on sensor data is proposed by [20]. This approach will serve as one of the baselines in the experimental section (as it is the only unsupervised approach on the sensor and not on log data). An approach [21] works on time series data in an unsupervised manner. It uses the LSTM-based autoencoder and is trained on the existing log data dataset. The proposed anomaly detector achieves the AUC (area under the receiver-operator characteristic curve) of 0.59. Although it works on a drastically different type of dataset (log data as opposed to system monitoring data), it is the closest existing work to the scope of the research presented in this paper. As we show later in the paper, we can achieve much better results than the one reported for the log data models [21] by deploying an unsupervised anomaly detection approach on system monitoring data on a per-node basis. Table 1 summarizes the most relevant approaches described in this section, focusing on the training set and temporal dependencies.

	Tabular data	Time series
Supervised	[49, 9]	[47, 48, 10]
Semi-supervised	[5, 6, 43, 22]	
Unsupervised	[19, 20]	[21]

Table 1: Summary of anomaly detection approaches on HPC systems

The novelty of this paper is, in relation to the existing works, threefold:

- it introduces an *unsupervised time-series based* anomaly detection model named RUAD;
- it proposes a deep learning architecture that captures *time dependency*;
- the approach is evaluated on a *large scale production* dataset with *real anomalies* – this is the largest scale evaluation ever conducted on this kind of problem, to the best of our knowledge.

3. Methodology

In this section, we describe the proposed approach for unsupervised anomaly detection. We do not directly introduce the proposed method (the LSTM autoencoder deep network) as we want to show how it is a significant extension to the current state-of-the-art; thus, we start by introducing three baseline methods, i) exponential smoothing (serving as the most basic method for comparison), ii) unsupervised clustering and iii) the dense autoencoder used in [5]. We then describe our approach in detail and highlight its key strengths (the unsupervised training regime and the explicit inclusion of the temporal dimension).

3.1. Node anomaly labeling

We aim to recognize the severe malfunctioning of a node that prevents it from executing regular compute jobs. This malfunctioning does not necessarily coincide with removing a node for the production, as reported by Nagios. In our discussions with system administrators of CINECA, we have concluded that the best proxy for node availability is the most critical state, as reported by Nagios. For this reason, we have created a *new label* called *node anomaly* that has a value 1 if any subsystem reported by Nagios reports a critical state. From these events (reported anomalies), we then filter out known false positive events based on reporting tests or configurations in Jira [50]. Jira logs are supplied by CINECA. The labels used in our previous work [5] do not apply to M100 as they were extensively used to denote nodes being removed from production for testing and calibration. In this work, we are examining the early period of the HPC machine life-cycle, when several rounds of re-configuration were performed, thus partially disrupting the normal production flow of the system. Comparing the two labelling strategies in table 2, we can see that the overlap between the two is minimal. Additionally, there are far fewer anomalies as reported by the *node anomaly* mainly because the M100 went through substantial testing periods in the first ten months of operation where nodes are marked as removed from production while still functioning normally. In the remainder of the paper, class 0 or class 1 will *always* refer to the value of *node anomaly* being 0 or 1 respectively. Normal data is all data where *node anomaly* has value 0 and *anomalies* are instances where *node anomaly* has value 1.

	Node anomaly	
	0	1
Removed from production: False	12 139 560	4 280
Removed form production: True	15 783	12

Table 2: Comparison between *removed from production* and *node availability*. The anomalies studied in this work (node availability) significantly differ (and are more reliable) from anomalies studied in previous works. The new labels also mark much fewer events as anomalous.

3.2. Reconstruction error and result evaluation

In this section we will formally express the theoretical background underpinning the practical approaches for anomaly detection described in the following sections. The problem of anomaly detection can be formally stated as a problem of training the model M that estimates the probability P that a sequence of vectors of length W ending at time t_0 represents an anomaly at time t_0. The formulation of the problem is summarized in the following equation:

$$M : \vec{x}_{t_0-W+1}, \cdots, \vec{x}_{t_0} \rightarrow P(\vec{x}_{t_0} \textit{ is an anomaly}). \tag{1}$$

Vector $\vec{x}_t$ collects all feature values at time t; the features are the sensor measurements collected from the computing nodes. W is the size of the past window that the model M takes as input. In practice, this means that we are looking for a model that given a time series of a certain time length (W) produces as output the probability that the final data point in the time series ($\vec{x}_{t_0}$) correspond to an anomalous point, rather than a normal one. If the model does not take past values into account - like the dense model implemented as a baseline [5] - and the window size W is 1, the problem can be simplified. For instance, the problem statement for the window W of size equal to 0 can be summarised as:

$$M : \vec{x}_{t_0} \rightarrow P(\vec{x}_{t_0} \textit{ is an anomaly}). \tag{2}$$

The crucial difference between Equations 1 and 2 is that the former takes as input entire time sequences, while the latter works on single data point, disregarding their past. This is one of the key differences between the model proposed in this manuscript (RUAD) which *does* take into account temporal sequences, and the SOTA based on previous works of Borghesi et al.[5], which takes as input single data points.

In the case of autoencoders, model M is composed of two parts: autoencoder (a neural network) and the anomaly score, which is computed using the reconstruction error of the autoencoder. The reconstruction error is calculated as the sum of the absolute difference between the output of model A and the normalized input value for each feature: $Error(t_0) = \sum_i^N |\hat{x}_i - x_i|$ where N is the number of features and $\hat{\vec{x}}_{t_0}$ is the output of the model A. The error is then normalized by dividing it by the maximum error on the training set: $Normalized\ error(t_0) = \frac{Error(t_0)}{max(Error(t))}$. Using the normalized error is then possible to estimate the probability that a certain data point ($\vec{x}_{t_0}$) is an anomaly, as summarized in the following equation:

$$P(\vec{x}_{t_0}\ is\ an\ anomaly) = \begin{cases} 1,\ if:\ Normalized\ error \geq 1, \\ Normalized\ error,\ otherwise \end{cases} \tag{3}$$

The probability is directly proportional to the normalized error, as a lower error indicates a low probability of the point being an anomaly, while larger values represent points that are very likely to be anomalous. The maximum probability is capped to 1, in order to maintain its meaning. Based on probability $P(\vec{x}_{t_0}\ is\ an\ anomaly)$, the classifier makes the prediction whether the sequence $\vec{x}_{t_0-W}, \cdots, \vec{x}_{t_0}$ belongs to class 1 (anomaly) of class 0 (normal operation). This prediction depends on a threshold T, which is a tunable parameter. The mechanism to decide whether a data point is anomalous or not depending on the threshold T, is summarized in the following equation:

$$Class(\vec{x}_{t_0}) = \begin{cases} 1,\ if:\ P(\vec{x}_{t_0}\ is\ an\ anomaly) \geq T, \\ 0,\ otherwise \end{cases} \tag{4}$$

If the normalized error is larger than (or equal to) T then the data point is classified as an anomaly with a probability equal to one, i.e., we are certain that the point is an anomaly; alternatively, the data point is classified as corresponding normal behaviour. However, this approach suffers from the fact that it is not trivial to decide which is the optimal threshold – different thresholds might lead to different outcomes (see [51] for an extensive analysis of this phenomenon in a similar anomaly detection setting). To avoid selecting a specific threshold T, we introduce the Receiver-Operator Characteristic curve (ROC curve) as a performance metric, a standard practice to assess the quality of an anomaly detection approach regardless of the threshold choice. It allows us to evaluate the performance of the classification approach for all possible decision thresholds [52]. The receiver-operator characteristic curve plots the true-positive rate in relation to the false-positive rate. The random decision represents a linear relationship between the two – for a classifier to make sense, the ROC curve needs to be above the diagonal line. For each specific point on the curve, the better classifier is the one whose ROC curve is above the other. The overall performance of the classifier can be quantitatively computed as the Area Under the ROC Curve (AUC); a classifier making random decisions has the AUC equal to 0.5. AUC scores below 0.5 designate classifiers that are worse than random choice. The best possible AUC score is 1, which is achieved by a classifier that would achieve a true-positive rate equal to 1 while having a false-positive rate equal to 0 (broadly speaking, this is only achievable on trivial datasets or very simple learning tasks).

AUC has been chosen as the main evaluation metric as it is among the most widely used metric in the anomaly detection domain [53]. In particular, it has been suggested that the AUC protects from pitfalls that are common to other relatively simpler metrics (e.g., pure accuracy detection rate or F-score measures) [54]. However, also added another metric to the experimental evaluation, namely the F-score, which combines precision and recall (the raw accuracy is noted to be very misleading when dealing with anomaly detection tasks rather than pure classification, due to the inherent imbalance among normal and abnormal classes).

3.3. Trivial baseline: exponential smoothing

Exponential smoothing is implemented as a trivial baseline comparison. It is a simple and computationally inexpensive method that detects rapid changes (jumps) in values. If the anomalies were simply rapid changes in values with no correlation between features, a simple exponential smoothing method would be able to discriminate them. Therefore, we chose exponential smoothing as a first baseline as it is computationally inexpensive and requires no training set. Additionally, if exponential smoothing performs poorly, this underlines that we are indeed solving a non-trivial anomaly detection problem, for which more powerful models are needed.

For the baseline, we choose to implement exponential smoothing per feature independently. Exponential smoothing for feature i at time t is calculated as:

$$\hat{x}_t^i = \alpha x_t^i + (1 - \alpha)\hat{x}_{t-1}^i, \forall i \in F \tag{5}$$

where $\hat{x}_t^i$ is an estimate of x^i at time t and α is a parameter of the method called the *smoothing factor* ($0 \leq \alpha \leq 1$), which governs the degree of smoothness of the resulting processed series. Larger values of α reduce the level of smoothing, and if $\alpha = 1$ the output series is equal to the observed one – in practice, smaller smoothing factors give more importance to past data points, while larger values focus on more recent data. We do this for all features in set F. The estimate at the beginning of the observation is equal to the actual value at time t_0: $\hat{x}_{t_0}^i = x_{t_0}^i$.

3.4. Unsupervised baseline: clustering

A possible approach to unsupervised anomaly detection is to use standard unsupervised machine learning techniques such as k-means clustering proposed by [20]. The clusters are determined on the train set; each new instance belonging to the test set is associated with one of the pre-trained clusters. We opted for this particular unsupervised technique for the comparison as it is the only unsupervised method found in the literature (to the best of our knowledge) which uses sensor data and not logs - and thus, we guarantee a fair comparison. It has to be noted, however, that clustering, while belonging to the field of unsupervised machine learning *cannot detect anomalies* in an unsupervised manner - for each of the clusters determined on the train set, the probability for the anomaly has to be calculated. This probability can only be calculated using the labels.

In this work, the clustering approach inspired by [20] is implemented to prove the validity of the obtained results. We have used K-means clustering [19] like it has been proposed in [20]. We have trained the clusters on the train set. Based on the silhouette score[1] on the train set, we have determined the optimal number of clusters for each node[2]. The percentage of instances that belong to class 1 is calculated for each of the determined clusters. We use this percentage of anomalous instances as the anomaly probability for each instance assigned to a specific cluster. The train and test set split is the same as in all other evaluated methods.

For the practical implementation of the clustering baseline, an anomaly probability has to be assigned to each cluster. To generate and associate this anomaly probability a *post-training* action should be performed by domain experts who should manually identify anomalous clusters of instances, making it a costly operation This analysis, however, cannot produce anomaly probability estimation that is better than the probability estimated from the actual labels. By analysing the actual labels (which are not available in real time in an actual deployment, as they can be obtained only via *post mortem* data analysis), we thus obtain an upper bound for the performance of the clustering method.

3.5. Semi-supervised baseline: dense autoencoder

The competitive baseline method is based on the current state-of-the-art dense autoencoder model proposed by [5]. Autoencoders are types of neural networks (NN) trained to reproduce their input. The network is split into two (most often symmetric) parts: encoder and decoder. The role of the encoder is to compress the input into a more condensed representation. This representation is called the *latent layer*. To prevent the network from learning a simple identity function, we choose the latent layer to be smaller than the original input size (number of input features) [6]. The role of the decoder is to reconstruct the original input using the latent representation.

Dense autoencoders are a common choice for anomaly detection since we can restrict their expressive power by acting on the size of the latent layer. Compressing the latent dimension forces the encoder to extract the most salient characteristics from the input data; unless the input data is highly redundant, the autoencoder cannot correctly learn to recreate its input after a certain latent size reduction. In the current state-of-the-art for anomaly detection in production supercomputers ([5]) the dense autoencoder is used in a semi-supervised fashion, meaning that the network is trained using only data points corresponding to the normal operation of the supercomputer nodes (Class 0). Semi-supervised training is doable as the normal points are the vast majority and thus are readily available; however, this requires having labelled data or at least a certainty that the HPC system was operating in normal conditions for a sufficiently long period of time. Once the autoencoder has been trained using only normal data, it will be able to recognize similar but previously unseen points. Conversely, it will struggle to reconstruct new points which do not follow the learned normal behaviour, that is, the anomalies we are looking for; hence, the reconstruction error will be higher. The structure of the autoencoder model is presented in Figure 1a. The dense autoencoder does not take into account the temporal dynamics of the data – its input and target output are the same vector (the set of metrics collected from a computing node). The input and output data for the SOTA anomaly detection approach is summarized in the following equation:

$$SOTA : \vec{x}_{t_0} \rightarrow \vec{x}_{t_0}. \tag{6}$$

This is the standard approach of autoencoder neural networks, which strive to reproduce their input as faithfully as possible.

3.6. Recurrent unsupervised anomaly detection: RUAD

Moving beyond the state-of-the-art model, we propose a different approach, RUAD. It takes as input a sequence of vectors and then tries to reconstruct only the *last vector in the sequence*. The input and output data for the proposed model *RUAD* is depicted in equation 7:

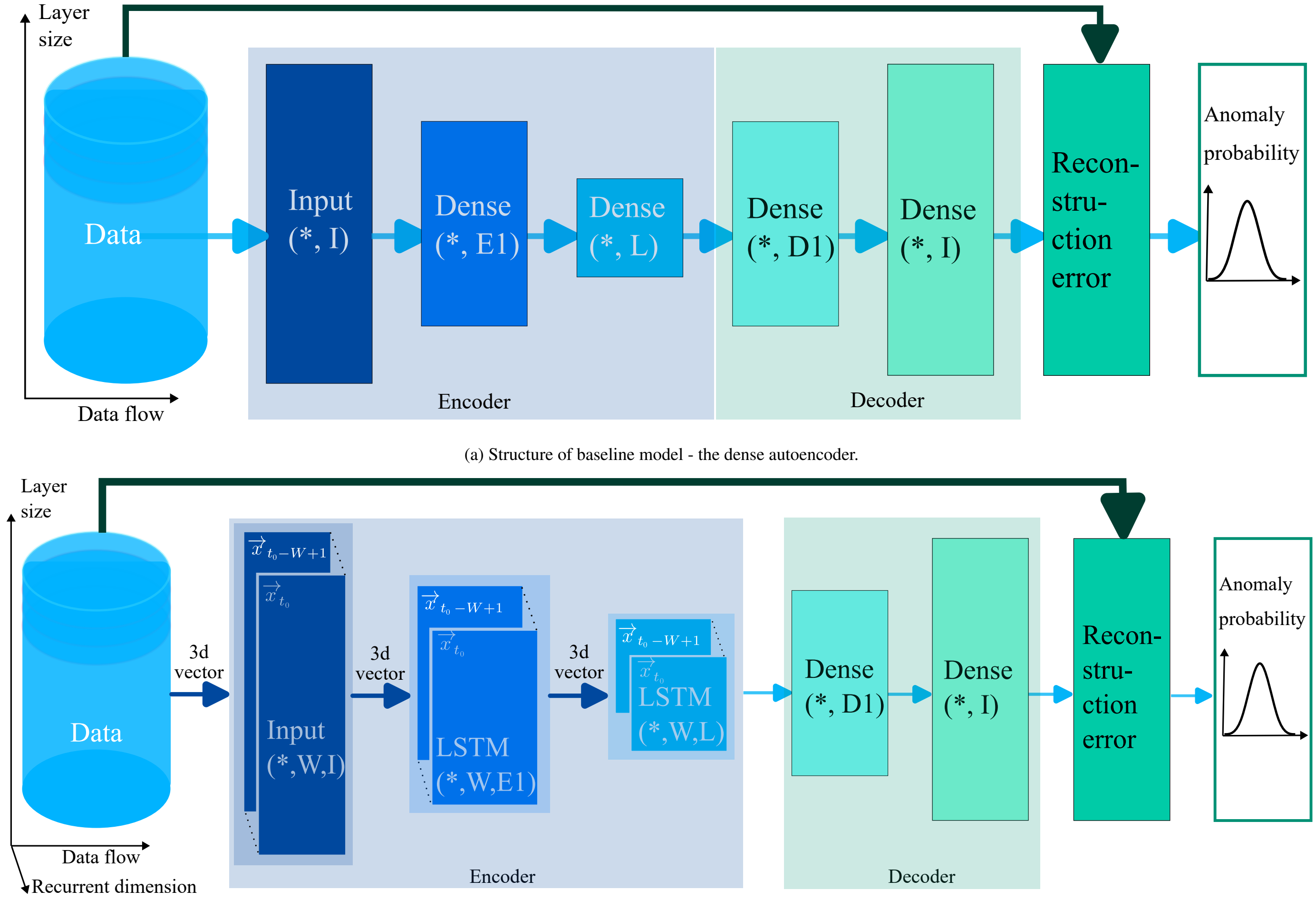
$$RUAD : \vec{x}_{t_0-W+1}, \cdots, \vec{x}_{t_0} \rightarrow \vec{x}_{t_0}. \tag{7}$$

The input sequence length is a tunable parameter that specifies the size of the observation window W. The idea of the proposed approach is similar to the dense autoencoder in principle, but with a couple of significant extensions: 1) we are encoding an input sequence into a more efficient representation (latent layer) and 2) we train the autoencoder in an unsupervised fashion (thus removing the requirement of labelled data). The key insight in the first innovation is that while the data describing supercomputing nodes is composed of multi-variate time series, the state-of-the-art does not explicitly consider the temporal dimension – the dense autoencoder has no notion of time nor of *sequence* of data points. To overcome this limitation, our approach works by encoding the sequence of values *leading up to the anomaly*. The encoder network is composed of Long Short-Term Memory (LSTM) layers, which have been often proved to be well suited to the context where the temporal dimension is relevant [56]. An LSTM layer consists of recurrent cells that have an input from the previous timestamp and from the long-term memory.

To address the scale of current pre-exascale and future exascale HPC systems that will consist of thousands of nodes [3], we want a scalable anomaly detection approach. The most scalable approach currently for anomaly detection on a whole supercomputer is a node-specific approach as each compute

[1] the Silhouette score is a measure of performance for a clustering method. It measures how similar an instance is to others in its own cluster compared to instances from the other clusters [55]. It is calculated as $S_{score} = \frac{b-a}{max(a,b)}$ where a is the mean inter-cluster distance, and b is the mean nearest cluster distance for each sample.

[2] Optimal number of clusters is the number of clusters that produces the highest silhouette score on the train set.



(a) Structure of baseline model - the dense autoencoder.

(b) Structure of the proposed RUAD model consisting of the LSTM encoder and dense decoder.

Figure 1: The proposed approach replaces the encoder of the baseline model (1a) with the LSTM autoencoder (1b). The last layer of LSTM encoder returns a vector (not a temporal sequence) which is then passed to the fully connected decoder. W is the window size, I is the size of the input data, L is the size of the latent layer and $E1$ and $D1$ are sizes of encoder and decoder layer respectively. Chosen parameters for L, W, $E1$ and $D1$ are listed in Section 4.3.

node can train its own model. Still, we want to achieve this by minimally impacting the regular operation of the HPC system. This is why it is important for the proposed solution to have a small overhead. Additionally, since we want to train a per-node model, we want the method to be data-efficient. To address these requirements, we choose not to make the decoder symmetric to the encoder. The proposed approach is thus comprised of a Dense decoder and an LSTM encoder. LSTM encoder output is passed into a dense decoder trained by reproducing the final vector in an input sequence. The decoder network is thus composed of fully connected dense layers. The architecture of the proposed approach is compared to the state-of-the-art approach in Figure 1.

The reduced complexity of training allows us to train a separate model for each compute node. As shown previously ([51]), node-specific models provide better results than a single model trained on all data. We decided to adopt this scheme (one model per node) after a preliminary empirical analysis showed no significant accuracy loss while the training time was vastly reduced (by approximately 50%); this is very important in our case as we trained one DL model for each of the nodes of Marconi 100 (980+), definitely a non-negligible computational effort.

3.7. *Data pre-processing*

As introduced in Section 3.6, our proposed methodology consists of training a model for each node. Figure 2 describes the whole data pre-processing pipeline. The data from each node is first split into training and test sets. The training set contains 80% of data, and the test set contains the remaining 20% of data (roughly the last two months of data – the data is split into time-consecutive chunks). It is important to stress that we have chosen to have two non-overlapping datasets for training and testing. This avoids the cross-transferring of information when dealing with sequencing. Moreover, the causality of the testing is preserved: the model is training without using data from the future, as no random split has been performed (as it is more common when not dealing with temporal data). This makes the results valid for in-practice usage.

For semi-supervised training, the training set is filtered by removing anomalous events (anomalous events are identified by the *node anomaly* label as described in Section 3.1). We name this filter the *semi-supervised filter*, as depicted in Figure

2. For unsupervised learning, the training set is not filtered, as both normal and anomalous points are indeed used. For both the cases (unsupervised and semi-supervised learning), labels are used to evaluate the results. After filtering, a scaler is fitted to training data. A scaler is a transformer that scales the data to the [0, 1] interval. In the experimental part, a min/max scaler is used on each feature [57]. After having been fitted on the training data, the scaler is applied to the test data – for rescaling the test set, min and max values of the training set are used (as it is standard practice in DL methods).

After scaling, both training and test sets are filtered out to ensure time consistency (as depicted in Figure 3). The time consistency filter consists of two steps: 1) the data is split into chunks of successive timestamps (in the case of semi-supervised training, successive timestamps contain no anomalies), 2) all chunks that are smaller than the input sequence of length W are dropped. Batches of sequences of length W are then generated for each chunk individually. Timestamps with missing values are dropped from the dataset (for all methods); missing timestamps – like anomalies – also split that dataset into smaller chunks. This – unlike anomalies – has no significant impact on the comparison of the methods as it is done for all analysed approaches.

Removing anomalous data points results in splitting the dataset into smaller chunks. As illustrated in Figure 3, all data between two anomalous data points closer together than the length of the input sequence W are removed from the training set. An important observation is that because we have to ensure time consistency, *removing anomalous data points results in also removing normal data points*. This, as discussed in the section 4.5, causes $RUAD$ (which is unsupervised) to slightly outperform $RUAD_{semi}$ (which is semi-supervised). $RUAD$ outperforms $RUAD_{semi}$ as it is trained on a larger dataset (the removal of anomalies and the connected normal points for time consistency reduces the training set of the latter). For larger W, more (normal) data points are dropped from the training set. This contributes to the decline in performance of $RUAD$ and $RUAD_{semi}$ with larger W (this is fully discussed in section 4.5). The average percentage (across all nodes in the dataset) of *normal data* that is removed, depending on the length of the input sequence, is collected in table 3.

Input sequnece length	Avreage percentage of removed normal data
5	4.7%
10	7.9%
20	13.8%
40	23.5%

Table 3: Average percentage of removed normal data due to semi-supervised and time consistency filters.

3.8. Summary of evaluated methods

We compare our proposed approach RUAD against established semi-supervised and unsupervised baselines. Summary of pre-processing filters is presented in Table 4. The semi-supervised filter is applied to all semi-supervised approaches. A time consistency filter is applied to methods that explicitly consider the temporal dimension of the data: Exponential smoothing and RUAD. RUAD and the current SOTA anomaly detection approach based on dense autoencoders ([5]) is evaluated in both semi-supervised and unsupervised versions.

	Filters		
Model	Semi-supervised	Time consistency	Name
Trivial baseline: exponential smoothing	NO	YES	EXP
Unsupervised baseline: clustering	NO	NO	CLU
DENSE autoencoder baseline semi-supervised	YES	NO	$DENSE_{semi}$
DENSE autoencoder baseline unsupervised	NO	NO	$DENSE_{un}$
RUAD semi-supervised	YES	YES	$RUAD_{semi}$
RUAD unsupervised	NO	YES	$RUAD$

Table 4: Short names and training strategies for examined methods. $DENSE_{semi}$ is the current SOTA [5].

Method	Training set required	Post-training
EXP	Unlabeled dataset	No action required
CLU [20]	Unlabeled dataset	Assigning anomaly probability to clusters
$DENSE_{semi}$[5]	Labeled dataset	No action required
$RUAD$	**Unlabeled dataset**	**No action required**

Table 5: Caparison of implemented approaches relating to the training set requirements.

We wish to highlight that, unlike the unsupervised learning baseline [20], our proposed method RUAD requires no additional action after the training of the model (like the manual analysis of the clusters). The approach RUAD, proposed in this work, works on an *unlabeled dataset* and requires no additional *post training analysis*. A summary of approaches relating to training set requirements is presented in Table 5.

4. Experimental results

4.1. Experimental setting

The focus of the experimental part of this work is Marconi 100 (M100) HPC system, located in the CINECA supercomputing centre. It is a tier-0 HPC system that consists of 980 compute nodes organized into three rows of racks. Each compute node has 32 core CPU, 256 GB of RAM and 4 Nvidia V100 GPUs. In this work, nodes of the HPC system will be considered independent. This is also in line with the current SOTA works [18, 6, 5] where anomaly detection is performed per node. Future works will investigate inter-node dependencies in the anomaly detection task.

The monitoring system in an HPC setting typically consists of hardware monitoring sensors, job status and availability monitoring, and server room sensors. In the case of M100, hardware monitoring is performed by Examon[14], and system availability is provided by system administrators[15]. This raw information provided by Nagios, however, contains many false-positive anomalies. For this reason, we have constructed a new anomaly label called *node anomaly* described in Section 3.1.

For each of the 980 nodes of M100, a separate dataset was created. Dataset details are explained in Section 4.2. $DENSE$ and $RUAD$ models were trained and evaluated on the node-specific training and test sets for each node. The training set consisted of the first eight months of system operation, and the test set comprised the remaining two months. Such testing split ensures a fair evaluation of the model as described in Section 4.2. For the baseline, the exponential smoothing operation (defined in equation (5)) was applied only over the test set (as the

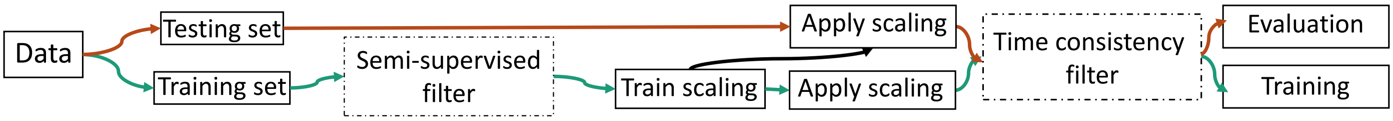


Figure 2: Data processing schema. The data flow is represented by green (training set) and orange (test set) lines. The scaler is trained on the training set and applied on test set to avoid contaminating the test set. Semi-supervised and time consistency filters are optional and applied only when required by the modeling approach as indicated in Table 4

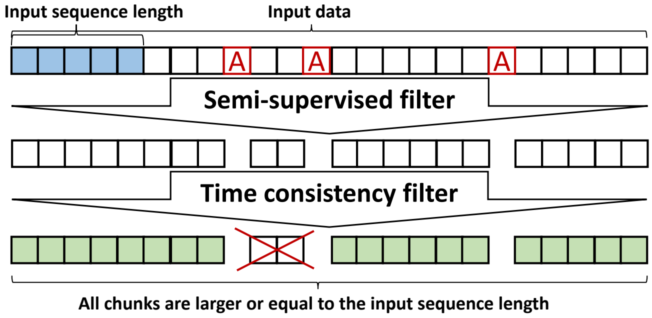


Figure 3: To ensure time consistency after removing the anomalous data (A in the Figure), we first split the data into chunks of successive timestamps without anomalies. Then we remove all chunks that are shorter than the input sequence length. Training sequences are only generated on the remaining chunks (green in the Figure).

approach requires no training). For each node, the scaler (for min and max scaling) was trained on training data and applied to test data. All results discussed in this section are *combined results from all 980 nodes of M100*.

The dense autoencoder and the RUAD model were trained in two different regimes: semi-supervised and unsupervised. For the semi-supervised training, the semi-supervised filter was applied that removed all data points corresponding to anomalies. In the unsupervised case, no such filtering was performed. It can hence be noticed one of the key advantages of the unsupervised approach: *no data pre-processing needs to be done and no preliminary knowledge about the computing nodes condition is required*.

For all three approaches (exponential smoothing, dense autoencoder and the *RUAD*), the probability for an anomaly (class 1) was estimated from reconstruction error as explained in Section 3.2. The probabilities from the test sets of all nodes from a single modelling approach (e.g. RUAD with observation window of length $W = 40$) were collected together to plot the Receiver Operator Characteristic (ROC) curve that is a characteristic for the modelling approach across all nodes. For clustering baseline and exponential smoothing (worst performing baselines), the ROC curve is compared against a dummy classifier which randomly chooses the class.

Source	Features
Hardware monitoring	ambient temp., dimm[0-15] temp., fan[0-7] speed, fan disk power, GPU[0-3] core temp. , GPU[0-3] mem temp. , gv100card[0-3], core[0-3] temp. , p[0-1] io power, p[0-1] mem power, p[0-1] power, p[0-1] vdd temp. , part max used, ps[0-1] input power, ps[0-1] input voltage, ps[0-1] output current, ps[0-1] output voltage, total power
System monitoring	CPU system, bytes out, CPU idle, proc. run, mem. total, pkts. out, bytes in, boot time, CPU steal, mem. cached, stamp, CPU speed, mem. free, CPU num., swap total, CPU user, proc. total, pkts. in, mem. buffers, CPU idle, CPU nice, mem. shared, PCIe, CPU wio, swap free

Table 6: An anomaly detection model is created only on hardware and application monitoring features. More granular information regarding individual jobs is not collected to ensure the privacy of the HPC system users.

4.2. Dataset

The dataset used in this work consists of a combination of information recorded by Nagios (the system administration tool used to visually check the health status of the computing nodes) and the Examon monitoring systems; the data encompasses the first ten months of operation of the M100 system. The procedure for obtaining a *node anomaly label* is described in Section 3.1. The features collected in the dataset are listed in table 6. The dataset contains 462 features. The data covers 980 compute nodes and five login nodes. Login nodes have the same hardware as the compute nodes but are reserved primarily for job submission and accounting. Thus we removed them from our analysis. The data is collected by the University of Bologna with approval from CINECA[3].

In order to align different sampling rates of different reporting services (each of the sensors used has a different sampling frequency), 15 minute aggregates of data points were created. 15 minute interval was chosen as it is the native sampling frequency of the Nagios monitoring service (where our labels come from). Four values were calculated for each 15 minute period and each feature: minimum, maximum, average, and variance.

[3] CINECA is a public university consortium and the main supercomputing centre in Italy[58].

4.3. Hyperparameters

Hyper-parameters for all methods discussed in this paper were determined based on initial exploration on the set of 50 nodes. Chosen parameters performed best on the test from the initial exploration nodes (they achieved the highest AUC score on the test set). Results from the initial exploration set are excluded from the results discussed further in the chapter. Tuned hyperparameters include the structure of the neural nets (number and size of layers) and the smoothing factor of the exponential smoothing:

- Exponential smoothing: smoothing factor $\alpha = 0.1$
- Clustering: hyper-parameter (number of clusters) is trained on a train set for each node independently.
- Dense autoencoder: Structure of the network consists of 5 layers of shapes: (*,462), (*,16), (*,8), (*16), (*462).
- RUAD (LSTM encoder, dense decoder): Structure of the network consists of 5 layers of shapes: (*,W,462), (*,W,16), (*,W,8), (*,16), (*,462). W is the length of the input sequence (observation window). Chosen input sequence lengths W were: $5, 10, 20, 40$.

4.4. Area under the curve (AUC)

The most basic baseline, exponential smoothing (EXP) is implemented to demonstrate that the anomalies we observe are not simply unexpected spikes in the data signal. Furthermore, exponential smoothing is applied to each feature independently of other features. As shown in Figure 4, exponential smoothing performs even worse than a dummy classifier (random choice). Poor performance of exponential smoothing shows that the anomalies we are searching for are more complex than simple jumps in values for a feature.

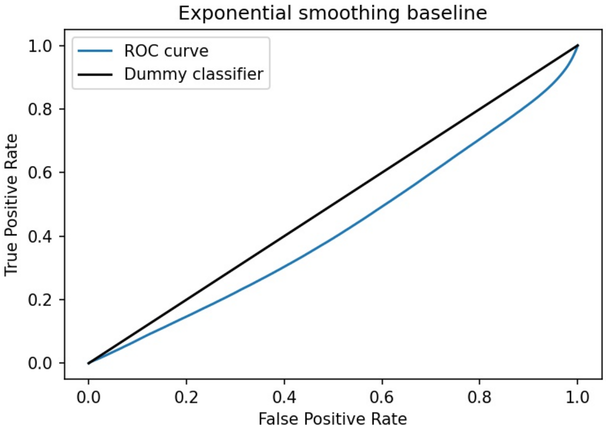


Figure 4: Combined ROC curve from all 980 nodes of M100 for the exponential smoothing baseline. Exponential smoothing performs even worse than the dummy classifier - anomaly detection based on exponential smoothing is completely unusable.

The simple clustering baseline performs better than the exponential smoothing baseline and better than the dummy classifier, as seen in Figure 5. However, as we will illustrate in the following sections, it performs worse than any other autoencoder method. This demonstrates that the problem we are addressing (anomaly detection on an HPC system) requires more advanced methodologies like semi-supervised and unsupervised autoencoders.

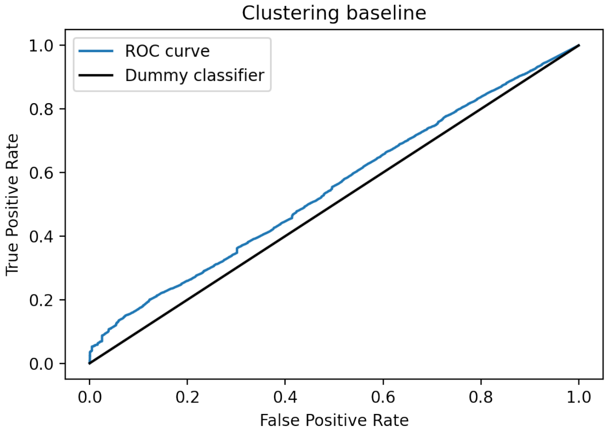


Figure 5: Combined ROC curve from all 980 nodes of M100 for the simple clustering baseline. This baseline performs only marginally better than the dummy classifier.

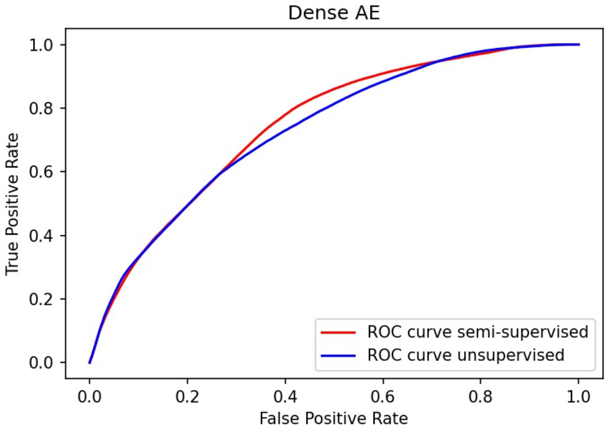


Figure 6: Combined ROC curve from all 980 nodes of M100 for the Dense autoencoder model. In the area interesting for practical application - True Positive Rate between 0.6 and 0.9 - semi-supervised approach outperforms unsupervised approach.

We now consider the dense autoencoder. We train a different network for each computing node of Marconi 100.

These parameters were identified after a (but clearly not complete) preliminary empirical evaluation, which suggested to us the best values; in addition, we restricted the initial parameter space by following the indications provided by previous works in the same domain

The network structure (number and size of layers) was determined during a thorough, preliminary exploration, which suggested to us the best values; in addition, we restricted the initial parameter space by following the indications provided by previous works in the same domain (Borghesi et al.[51]).

In line with the existing work[5], the semi-supervised learning approach $DENSE_{semi}$ slightly outperforms the unsupervised learning approach $DENSE_{un}$ as seen in Figure 6. The improved performance of the semi-supervised model is due to the peculiarities of the autoencoder network, namely its capability to reconstruct its input. For instance, let consider the unsupervised case. If anomalous examples are fed as input to the unsupervised model during the training phase, the autoencoder will learn to represent them in its latent space. Thus, it will be capable to partially reconstruct them as well, albeit with much more difficulty, as these anomalous points are extremely rare (the scarcity of anomalies is a critical assumption for the correct functioning of the unsupervised model). In turn, this implies that the unsupervised model will be slightly less capable of differentiating normal and anomalous points than the semi-supervised model, as the latter has been strictly fed only with normal examples.

This section examines the experimental results obtained with the RUAD model (unsupervised LSTM autoencoder). The most important parameter is the length of the input sequence W that is passed to the model. This parameter encodes our expectation of the length of the dependencies within the data. Since each data point represents 15 minutes of node operation, the actual period we observe consists of $W \times 15$ *min*. In this set of experiments, we selected the following time window sizes: 5 (75 minutes), 10 (2h30), 20 (5h), 40 (10h). These period lengths were obtained after a preliminary empirical evaluation; moreover, these time frames are in line with the typical duration of HPC workloads, which tend to span between dozens of minutes to a few hours[59]. We have trained the model in both semi-supervised $RUAD_{semi}$ and unsupervised $RUAD$ fashion for each selected window length. Results across all the nodes are collected in Figure 7.

4.5. Comparison of all approaches

The main metric for evaluating model performance is the area under the ROC curve (AUC). This metric estimates the classifiers' overall performance without the need to set a discrimination threshold [52]. The closer the AUC value is to 1, the better the classifier performs. AUC scores for implemented methods are collected in table 7. From the lower table in table 7 (rows correspond to different training regimes and columns to window size for $RUAD$ network) and upper table in 7 (rows correspond to the performance of different implemented baselines), we see that the proposed approach outperforms the existing baselines. The highest AUC achieved by the previous baselines is 0.7470 (achieved by the $DENSE_{semi}$. This is outperformed by $RUAD$ for *all window sizes*. The best performance of $RUAD$ is achieved by selecting the windows size 10 where it achieves an AUC of 7.672. This result clearly shows that some temporal dynamics contribute to the appearance of anomalies.

The final consideration is the impact of observation window length W on the performance of the RUAD model. One might expect that considering longer time sequences would bring benefits, as more information is provided to the model to recreate the time series. This is, however, not the case (as seen in table 7) as the $RUAD$ achieves the best performance of 0.7672 with window size 10. The performance then reduces sharply with window size 40, only achieving an AUC of 0.7473. Several factors might explain this phenomenon. For instance, in tens of hours, the workload on a given node might change drastically. Considering longer time series might thus force the RUAD model to concentrate on multiple workloads, hindering its learning task. Finally, an issue stems from the fact that there are gaps (periods of missing measurements) in the collected data (a very likely problem in many real-world scenarios). Longer sequences mean that more data has to be cut from the training set to ensure time-consistent sequences; this is because we are not applying gap-filling techniques at the moment[4], thus, sub-sequences missing some points need to be removed from the data set. Combining these two factors contributes to the model's decline in performance with longer observation periods.

Considering all discussed factors, the optimal approach is to use the proposed model architecture with window size $W = 10$ (i.e. 2 hours and 30 minutes), trained in an *unsupervised* manner. This configuration outperforms semi-supervised $RUAD_{semi}$ as well as the dense autoencoder. As mentioned in the related work (Section 2), labelled datasets are expensive to obtain in the HPC setting. Good unsupervised performance is why this result is promising - it shows us that if the anomalies represented a small fraction of all data, we could train an anomaly detection model even on an unlabeled dataset (in an unsupervised manner). Such a model not only achieves the state-of-the-art performance but *outperforms* semi-supervised approaches. The best AUC, achieved by the previous SOTA $DENSE_{semi}$, is 0.7470. The best AUC score achieved by $RUAD$ is 0.7672. Moreover, unsupervised training makes this anomaly detection model more applicable to a typical HPC (or even datacentre) system.

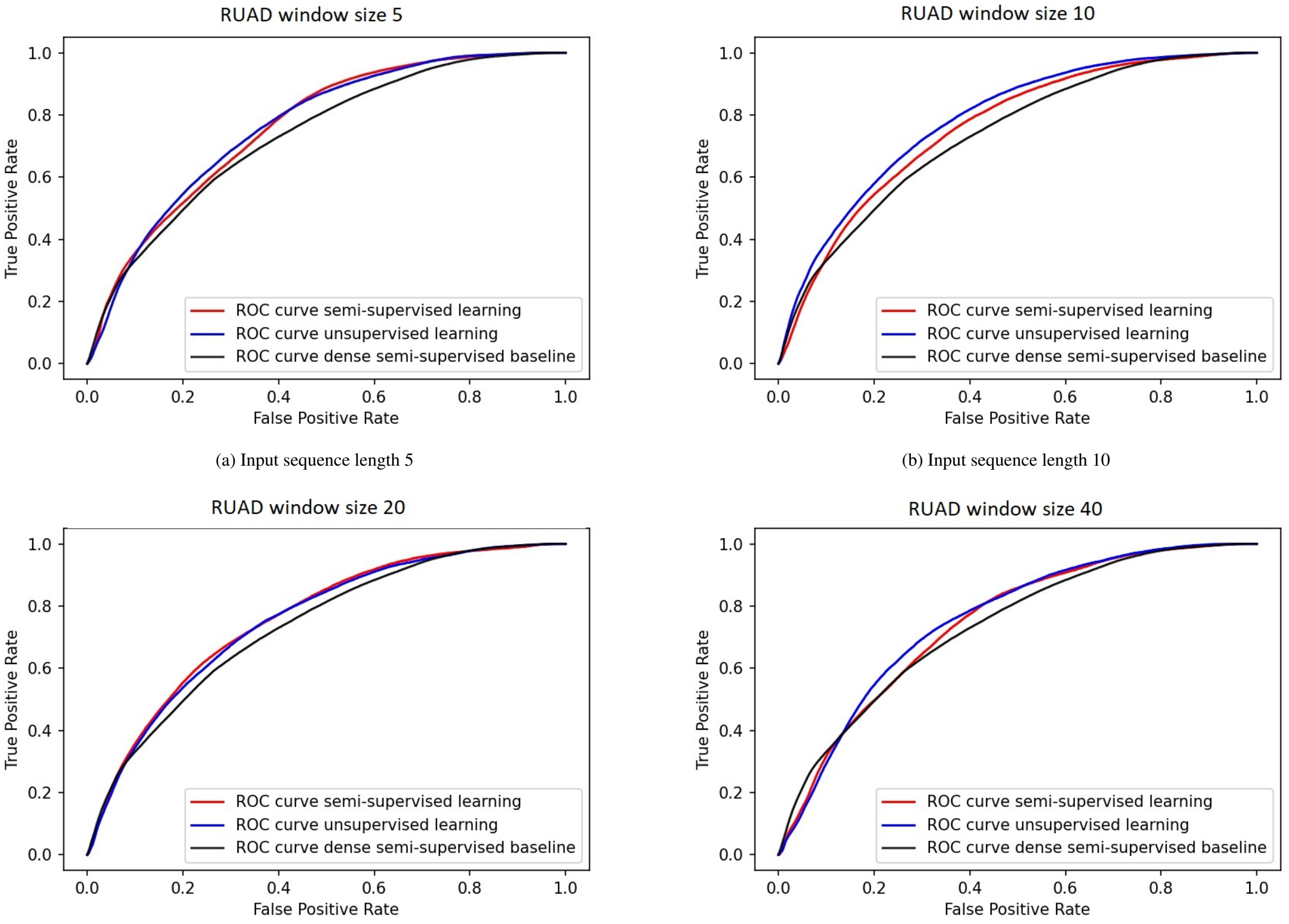
Compared to the previous SOTA for completely unsupervised anomaly detection CLU, which achieves an AUC of just 0.5478 $RUAD$ achieves significantly better results with an AUC 0f 0.7672. $RUAD$ thus sets a new SOTA for *unsupervised anomaly detection.*

Method	Combined AUC score
EXP	0.4276
CLU	0.5478
$DENSE_{semi}$	0.7470
$DENSE_{un}$	0.7344

Method	Combined AUC score			
Input sequence length	5	10	20	40
$RUAD_{semi}$	0.7632	0.7582	0.7602	0.7446
$RUAD$	**0.7651**	**0.7672**	**0.7655**	**0.7473**

Table 7: According to expectations, the semi-supervised dense autoencoder outperforms the unsupervised dense one (highlighted by the higher AUC score). $RUAD$ and $RUAD_{semi}$ *outperform all previous baselines*. In contrast to the dense autoencoders, the proposed approach $RUAD$ performs best in *unsupervised manner*.

[4] We decided not to consider such techniques for the moment, as we wanted to focus on the modelling approach and gap-filling methods tend to require additional assumptions and to introduce noise in the data.



(a) Input sequence length 5

(b) Input sequence length 10

(c) Input sequence length 20

(d) Input sequence length 40

Figure 7: Combined results from all 980 nodes of M100. Comparison of different window lengths for the RUAD model. For all window lengths, performances of semi-supervised and unsupervised approaches are similar. Performance of the proposed model (red and blue line) is compared to the state-of-the-art baseline semi-supervised autoencoder proposed by Borghesi et al.[5].

4.6. F1 scores

To illustrate the performance of the proposed classifiers, we present the comparison of F1 scores of all the methods with different thresholds in table 8. The threshold 1 represents a trivial classifier that classifies all data as normal operation. Scores lower than the scores of this trivial classifier represent classifiers that are useless for practical evaluation and should be excluded from the analysis (in table 8 are greyed out). For all autoencoder-based approaches ($RUAD$, $RUAD_{semi}$, $DENSE_{un}$, $DENSE_{semi}$) the difference in F1 score is small for different thresholds. This suggests that the predicted probabilities are either very low (for normal operation) or very high (for anomalies). The difference between semi-supervised training and unsupervised training is less noticeable but $RUAD$ still *outperforms all other approaches*. Particularly interesting is the comparison to the unsupervised benchmark CLU that, for the chosen thresholds, performs particularly bad – even worse than exponential smoothing.

5. Conclusions

The paper presents an anomaly detection approach for HPC systems (RUAD) that outperforms the current state-of-the-art approach based on the dense autoencoders [5]. Improving upon state-of-the-art is achieved by deploying a neural network architecture that considers the temporal dependencies within the data. The proposed model architecture achieves the highest AUC of 0.77 compared to 0.75, which is the highest AUC achieved by the dense autoencoders (on our dataset).

Another contribution of this paper is that the proposed method – unlike the previous work [5, 16, 17, 8] – achieves the best results in an *unsupervised training* case. Unsupervised training is instrumental as it offers a possibility of deploying an anomaly detection model to the cases where (accurately) labelled dataset is unavailable. The only stipulation for the deployment of *unsupervised* anomaly detection models is that the anomalies are rare – in our work, the anomalies accounted for only 0.035% of the data. The necessity to have a few anomalies in the training set, however, is not a significant limitation as HPC systems are already highly reliable machines

				F1 score									
			Threshold	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Method	EXP			0.009	0.767	0.893	0.927	0.941	0.947	0.948	0.948	0.948	0.948	0.894
	CLU			0.009	0.887	0.891	0.895	0.898	0.899	0.899	0.898	0.895	0.895	0.894
	$DENSE_{semi}$			0.009	0.465	0.941	0.952	0.953	0.953	0.953	0.953	0.953	0.953	0.894
	$DENSE_{un}$			0.009	0.298	0.926	0.951	0.952	0.953	0.953	0.953	0.953	0.953	0.894
	$RUAD_{semi}$	Input sequence length	5	0.009	0.391	0.936	0.957	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			10	0.009	0.395	0.935	0.957	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			20	0.009	0.420	0.937	0.957	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			40	0.009	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.894
	$RUAD$		5	0.009	0.333	0.928	0.956	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			10	0.009	0.307	0.924	0.957	0.959	0.960	0.960	0.960	0.960	0.960	0.894
			20	0.009	0.439	0.938	0.958	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			40	0.009	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.894

Table 8: Combined F1 scores for all compute nodes. F1 scores worse than the trivial classifier (decision threshold 1) are greyed out. RUAD outperforms all previous approaches, including the previous state-of-the-art ($DENSE_{semi}$ and $DENSE_{un}$).

with low anomaly rates [60, 1].

To illustrate the capabilities of the approach proposed in this work, we have collected an extensive and accurately labelled dataset describing the first 10 months of operation of the Marconi100 system in CINECA [58]. The creation of *accurately labelled* dataset was necessary to compare the performance of different models on the data rigorously. Because of the high z and large scale of the available dataset, we can conclude that for the model proposed in the paper, the unsupervised model *outperforms* semi-supervised model *even if accurate anomaly labels are available*. This is the first experiment of this type and magnitude conducted on a real, in-production datacentre (both in terms of the number of computing nodes considered and the length of the observation period).

In future works, we will further explore the problem of anomaly detection in HPC systems, in particular, discovering the root causes of the anomalies - e.g., *why* a computing node is entering a failure state? We also have plans to further extend and refine the collected dataset and make it available to the public, in accordance with the facility owners and regulations about users' personal data (albeit not considered in this work, information about the users submitting the jobs to the HPC system can indeed be collected).

6. Acknowledgments

This research was partly supported by the EuroHPC EU PILOT project (g.a. 101034126), the EuroHPC EU Regale project (g.a. 956560), EU H2020-ICT-11-2018-2019 IoTwins project (g.a. 857191), and EU Pilot for exascale EuroHPC EUPEX (g. a. 101033975). We also thank CINECA for the collaboration and access to their machines and Francesco Beneventi for maintaining Examon.

References

[1] W. Shin, V. Oles, A. M. Karimi, J. A. Ellis, F. Wang, Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–14. doi:10.1145/3458817.3476188.
URL `https://doi.org/10.1145/3458817.3476188`

[2] D. Milojicic, P. Faraboschi, N. Dube, D. Roweth, Future of hpc: Diversifying heterogeneity, in: 2021 Design, Automation Test in Europe Conference Exhibition (DATE), 2021, pp. 276–281. doi:10.23919/DATE51398.2021.9474063.

[3] A. Netti, W. Shin, M. Ott, T. Wilde, N. Bates, A conceptual framework for hpc operational data analytics, in: 2021 IEEE International Conference on Cluster Computing (CLUSTER), 2021, pp. 596–603. doi:10.1109/Cluster48925.2021.00086.

[4] L. A. Parnell, D. W. Demetriou, V. Kamath, E. Y. Zhang, Trends in high performance computing: Exascale systems and facilities beyond the first wave, in: 2019 18th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2019, pp. 167–176. doi:10.1109/ITHERM.2019.8757229.

[5] A. Borghesi, M. Molan, M. Milano, A. Bartolini, Anomaly detection and anticipation in high performance computing systems, IEEE Transactions on Parallel and Distributed Systems 33 (4) (2022) 739–750. doi:10.1109/TPDS.2021.3082802.

[6] A. Borghesi, A. Bartolini, et al., Anomaly detection using autoencoders in hpc systems, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 24–32.

[7] A. Borghesi, M. Milano, L. Benini, Frequency assignment in high performance computing systems, in: International Conference of the Italian Association for Artificial Intelligence, Springer, 2019, pp. 151–164.

[8] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, A. Borghesi, A machine learning approach to online fault classification in hpc systems, Future Generation Computer Systems (2019).

[9] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, A. Borghesi, Online fault classification in hpc systems through machine learning, in: European Conference on Parallel Processing, Springer, 2019, pp. 3–16.

[10] M. Du, F. Li, G. Zheng, V. 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, Association for Computing Machinery, New York, NY, USA, 2017, p. 1285–1298. doi:10.1145/3133956.3134015.
URL `https://doi.org/10.1145/3133956.3134015`

[11] F. Iannone, G. Bracco, C. Cavazzoni, et al., Marconi-fusion: The new high performance computing facility for european nuclear fusion modelling, Fusion Engineering and Design 129 (2018) 354–358.

[12] N. Beske, Ug3.2: Marconi100 userguide, accessed: 2020-08-17 (2020).
URL `https://wiki.u-gov.it/confluence/pages/viewpage.action?pageId=336727645`

[13] Top500list, https://www.top500.org/lists/top500/2020/06/ (2020).

[14] A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, C. Cavazzoni, Paving the way toward energy-aware and automated datacentre, in: Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–8. doi:10.1145/3339186.3339215.
URL `https://doi.org/10.1145/3339186.3339215`

[15] W. Barth, Nagios: System and network monitoring, No Starch Press, 2008.

[16] M. Molan, A. Borghesi, F. Beneventi, M. Guarrasi, A. Bartolini, An explainable model for fault detection in hpc systems, in: H. Jagode, H. Anzt, H. Ltaief, P. Luszczek (Eds.), High Performance Computing, Springer International Publishing, Cham, 2021, pp. 378–391.

[17] O. Tuncer, E. Ates, Y. e. a. et Zhang, Online diagnosis of performance variation in hpc systems using machine learning, IEEE Transactions on Parallel and Distributed Systems (9 2018).

[18] A. Netti, Z. Kiziltan, et al., Finj: A fault injection tool for hpc systems, in: European Conference on Parallel Processing, Springer, 2018, pp. 800–812.

[19] M. Dani, H. Doreau, S. Alt, K-means application for anomaly detection and log classification in hpc, in: Lecture Notes in Computer Science book series (LNAI,volume 10351), 2017, pp. 201–210. doi:10.1007/978-3-319-60045-1_23.

[20] A. Morrow, E. Baseman, S. Blanchard, Ranking anomalous high performance computing sensor data using unsupervised clustering, in: 2016 International Conference on Computational Science and Computational Intelligence (CSCI), 2016, pp. 629–632. doi:10.1109/CSCI.2016.0124.

[21] S. Bursic, A. D'Amelio, V. Cuculo, Anomaly detection from log files using unsupervised deep learning (09 2019).

[22] A. Borghesi, A. Libri, et al., Online anomaly detection in hpc systems, in: 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems, IEEE, 2019, pp. 229–233.

[23] G. Moschini, R. Houssou, J. Bovay, S. Robert-Nicoud, Anomaly and fraud detection in credit card transactions using the arima model (2020). arXiv:2009.07578.

[24] M. Ahmed, A. N. Mahmood, M. R. Islam, A survey of anomaly detection techniques in financial domain, Future Generation Computer Systems 55 (2016) 278–288. doi:https://doi.org/10.1016/j.future.2015.01.001.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X15000023`

[25] K. B. Lee, S. Cheon, C. O. Kim, A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes, IEEE Transactions on Semiconductor Manufacturing 30 (2) (2017) 135–142.

[26] L. Rosa, T. Cruz, M. B. de Freitas, P. Quitério, J. Henriques, F. Caldeira, E. Monteiro, P. Simões, Intrusion and anomaly detection for the next-generation of industrial automation and control systems, Future Generation Computer Systems 119 (2021) 50–67. doi:https://doi.org/10.1016/j.future.2021.01.033.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X21000431`

[27] I. Martins, J. S. Resende, P. R. Sousa, S. Silva, L. Antunes, J. Gama, Host-based ids: A review and open issues of an anomaly detection system in iot, Future Generation Computer Systems 133 (2022) 95–113. doi:https://doi.org/10.1016/j.future.2022.03.001.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X22000760`

[28] F. Cauteruccio, L. Cinelli, E. Corradini, G. Terracina, D. Ursino, L. Virgili, C. Savaglio, A. Liotta, G. Fortino, A framework for anomaly detection and classification in multiple iot scenarios, Future Generation Computer Systems 114 (2021) 322–335. doi:https://doi.org/10.1016/j.future.2020.08.010.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X19335253`

[29] R. Xu, Y. Cheng, Z. Liu, Y. Xie, Y. Yang, Improved long short-term memory based anomaly detection with concept drift adaptive method for supporting iot services, Future Generation Computer Systems 112 (2020) 228–242. doi:https://doi.org/10.1016/j.future.2020.05.035.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X20302235`

[30] S. Fu, S. Zhong, L. Lin, M. Zhao, A re-optimized deep autoencoder for gas turbine unsupervised anomaly detection, Engineering Applications of Artificial Intelligence 101 (2021) 104199. doi:https://doi.org/10.1016/j.engappai.2021.104199.
URL `https://www.sciencedirect.com/science/article/pii/S0952197621000464`

[31] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, N. V. Chawla, A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data, CoRR abs/1811.08055 (2018). arXiv:1811.08055.

[32] P. V. Astillo, D. G. Duguma, H. Park, J. Kim, B. Kim, I. You, Federated intelligence of anomaly detection agent in iotmd-enabled diabetes management control system, Future Generation Computer Systems 128 (2022) 395–405. doi:https://doi.org/10.1016/j.future.2021.10.023.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X21004192`

[33] T. Salman, D. Bhamare, A. Erbad, R. Jain, M. Samaka, Machine learning for anomaly detection and categorization in multi-cloud environments, 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud) (2017). arXiv:1812.05443, doi:10.1109/CSCloud.2017.15.

[34] M. Molan, Pre-processing for Anomaly Detection on Linear Accelerator. CERN openlab online summer intern project presentations (Sep 2020).

[35] M. Gamell, K. Teranishi, et al., Modeling and simulating multiple failure masking enabled by local recovery for stencil-based applications at extreme scales, IEEE Transactions on Parallel and Distributed Systems 28 (10) (2017).

[36] E. Meneses, X. Ni, et al., Using migratable objects to enhance fault tolerance schemes in supercomputers, IEEE Transactions on Parallel and Distributed Systems 26 (7) (2015) 2061–2074. doi:10.1109/TPDS.2014.2342228.

[37] I. Boixaderas, D. Zivanovic, et al., Cost-aware prediction of uncorrected dram errors in the field, in: 2020 SC20: International Conference for HPC, Networking, Storage and Analysis (SC), IEEE Comp. Soc., Los Alamitos, CA, USA, 2020, pp. 1–15.

[38] G. Iuhasz, D. Petcu, Monitoring of exascale data processing, in: 2019 IEEE International Conference on Advanced Scientific Computing (ICASC), 2019, pp. 1–5. doi:10.1109/ICASC48083.2019.8946279.

[39] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, et al., Exascale computing study: Technology challenges in achieving exascale systems, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15 (2008).

[40] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, ACM Comput. Surv. (mar 2020). doi:10.1145/3439950.

[41] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, ACM Comput. Surv. 54 (2) (Mar. 2021). doi:10.1145/3439950.
URL `https://doi.org/10.1145/3439950`

[42] G. Lemaître, F. Nogueira, C. K. Aridas, Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning, Journal of Machine Learning Research 18 (17) (2017) 1–5.

[43] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, Engineering Applications of Artificial Intelligence 85 (2019) 634–644.

[44] P. Wu, C. A. Harris, G. Salavasidis, A. Lorenzo-Lopez, I. Kamarudzaman, A. B. Phillips, G. Thomas, E. Anderlini, Unsupervised anomaly detection for underwater gliders using generative adversarial networks, Engineering Applications of Artificial Intelligence 104 (2021) 104379. doi:https://doi.org/10.1016/j.engappai.2021.104379.
URL `https://www.sciencedirect.com/science/article/pii/S095219762100227X`

[45] O. Tuncer, E. Ates, et al., Diagnosing performance variations in hpc applications using machine learning, in: International Supercomputing Conference, Springer, 2017, pp. 355–373.

[46] B. Aksar, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, E2ewatch: An end-to-end anomaly diagnosis framework for production hpc systems, in: European Conference on Parallel Processing, Springer, 2021, pp. 70–85.

[47] B. Aksar, Y. Zhang, E. Ates, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems, in: B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, P. Luszczek (Eds.), High Performance Computing, Springer International Publishing, Cham, 2021, pp. 195–214.

[48] E. Baseman, S. Blanchard, N. DeBardeleben, A. Bonnie, A. Morrow, Interpretable anomaly detection for monitoring of high performance com-

puting systems, in: Outlier Definition, Detection, and Description on Demand Workshop at ACM SIGKDD. San Francisco (Aug 2016), 2016, pp. 1–27.

[49] B. Aksar, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, E2ewatch: An end-to-end anomaly diagnosis framework for production hpc systems, in: L. Sousa, N. Roma, P. Tomás (Eds.), Euro-Par 2021: Parallel Processing, Springer International Publishing, Cham, 2021, pp. 70–85.

[50] Wikipedia, Jira (software) — Wikipedia, the free encyclopedia, http://en.wikipedia.org/w/index.php?title=Jira\%20(software)&oldid=1052315603, [Online; accessed 04-December-2021] (2021).

[51] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, Engineering Applications of Artificial Intelligence 85 (2019) 634–644. doi:https://doi.org/10.1016/j.engappai.2019.07.008.
URL https://www.sciencedirect.com/science/article/pii/S0952197619301721

[52] Receiver operating characteristic (Nov 2021).
URL https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[53] S. Kim, K. Choi, H. Choi, B. Lee, S. Yoon, Towards a rigorous evaluation of time-series anomaly detection, CoRR abs/2109.05257 (2021). arXiv:2109.05257.
URL https://arxiv.org/abs/2109.05257

[54] D. Fourure, M. U. Javaid, N. Posocco, S. Tihon, Anomaly detection: How to artificially increase your f1-score with a biased evaluation protocol, CoRR abs/2106.16020 (2021). arXiv:2106.16020.
URL https://arxiv.org/abs/2106.16020

[55] K. R. Shahapure, C. Nicholas, Cluster quality analysis using silhouette score, in: 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), 2020, pp. 747–748. doi:10.1109/DSAA49011.2020.00096.

[56] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, M. Weyrich, A survey on long short-term memory networks for time series prediction, Procedia CIRP 99 (2021) 650–655.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[58] Wikipedia, CINECA — Wikipedia, the free encyclopedia, http://en.wikipedia.org/w/index.php?title=CINECA&oldid=954269846, [Online; accessed 04-December-2021] (2021).

[59] M. C. Calzarossa, L. Massari, D. Tessera, Workload characterization: A survey revisited, ACM Computing Surveys (CSUR) 48 (3) (2016) 1–43.

[60] J. Dongarra, Report on the fujitsu fugaku system, University of Tennessee-Knoxville Innovative Computing Laboratory, Tech. Rep. ICLUT-20-06 (2020).