The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-023-11255-x

ORIGINAL ARTICLE

Anomaly detection and virtual reality visualisation in supercomputers

David Mulero-Pérez[1] · Manuel Benavent-Lledó[1] · Jorge Azorín-López[1] · Diego Marcos-Jorquera[1] · José García-Rodríguez[1]

Received: 3 October 2022 / Accepted: 10 March 2023

Abstract
Anomaly detection is the identification of events or observations that deviate from the expected behaviour of a given set of data. Its main application is the prediction of possible technical failures. In particular, anomaly detection on supercomputers is a difficult problem to solve due to the large scale of the systems and the large number of components. Most research works in this field employ machine learning methods and regression models in a supervised fashion, which implies the need for a large amount of labelled data to train such systems. This work proposes the use of autoencoder models, allowing the problem to be approached with semi-supervised learning techniques. Two different model training approaches are compared. The former is a model trained with data from all the nodes of a supercomputer. In the latter approach, observing significant differences between nodes, one model is trained for each node. The results are analysed by evaluating the positive and negative aspects of each approach. On the other hand, a replica of the Marconi 100 supercomputer is developed in a virtual reality environment that allows the data from each node to be visualised at the same time.

Keywords Anomaly detection · Virtual reality and Machine learning · Supercomputing

1 Introduction

Industry 4.0 is the fourth major industrial stage that has taken place since the beginning of the Industrial Revolution. It is characterised by a fusion of technologies currently being tested or developed. There is a trend towards automation and data exchange, which is very important in the context of manufacturing and development technologies. This includes technologies such as cyber-physical systems, the Internet of Things (IoT) and High-Performance Computing (HPC). HPC or Parallel Computing that refers to individual computer systems in a cluster as nodes that can perform complex computations on PCs, Microcontroller Units (MCUs) and Edge devices with multi-core Central Processing Units (CPUs) and Graphics Processing Units (GPUs).

The use of machine learning and, specifically, deep learning techniques is becoming increasingly common, both in the improvement of industrial production processes and in their preventive maintenance. We find examples such as deep-learning-based visual control assembly assistant [1]. This work enables real-time evaluation of the activities in the assembly process to identify errors. In [2], the optical quality control process in the printing industry has been improved. To do this, they design deep learning models that allow them to classify samples according to their quality. This is also a very resource-intensive computer vision problem, as it involves training models with very high-resolution images.

With the advent of Industry 4.0 and the use of more computationally demanding techniques, heterogeneous high-performance systems are coming to the fore. This technology is used as a driving force to deploy various industrial applications. Angelopoulos et al. [3] discuss the emergence of these methods in different fields. In the manufacturing industry, HPC is used to support product design, optimisation or testing. This technology is increasingly used in different types of industries, highlighting the automotive industry that has become more widespread [4].

✉ David Mulero-Pérez
dmulero@dtic.ua.es

Manuel Benavent-Lledó
mbenavent@dtic.ua.es

Jorge Azorín-López
jazorin@dtic.ua.es

Diego Marcos-Jorquera
dmarcos@dtic.ua.es

José García-Rodríguez
jgarcia@dtic.ua.es

[1] Department of Computer Science and Technology, University of Alicante, Alicante, Spain

Published online: 22 May 2023

In some cases, it requires an environment that runs complex codes in real-time and with dynamic adjustment of resources for workloads. This is known as on-demand HPC and requires the HPC system to be constantly highly available, otherwise the processing may not be performed in time. Another important factor is the power consumption and the types of HPC architectures to be used. The current trend in HPC is towards heterogeneous platforms and mixed programming models. HPC systems have to cope with different performance targets that may vary from time to time depending on the overall system requirements. Supercomputing is the heart of AI and Big Data, coupled with 5-G Networks together creates Artificial Intelligence of Things (AIoT) that forms the backbone of the high-speed, smart, connected networks [5]. The use of these technologies together brings an improvement in economic performance and efficiency for the industrial sector.

Preventive maintenance and early anomaly detection are necessary to ensure the proper performance of these systems. In data science, anomaly detection is normally understood as the identification of outliers, events or observations which deviate significantly from the majority of the data. In our case, analyzing time-series data in real-time, identifying unexpected values of node sensors, which differ from the normal data (no anomalies). One of the problems of working with this type of data is that it can be an extreme case of an unbalanced supervised learning problem, the vast majority of data generated by real supercomputers is, by definition, normal, in our case.

Currently, the issues concerning the High Performance Computing systems maintenance are increasing by rapidly becoming larger and complex. Anomaly identification can assist with pinpointing where a error happens, upgrading root cause analysis and rapidly getting technical support prior to arriving at a critical state.

Anomaly detection within large systems is a problem that is becoming increasingly important as more and more large-scale computing centres emerge. It is used to predict possible incidents or technical failures. In the case of supercomputers, this is a difficult problem to solve due to the cost of generating labelled data and classical methods require a large amount of data to train the models. In this paper, several autoencoder neural network models are implemented and evaluated, which allow the problem to be dealt by using semi-supervised learning techniques. We focus on the Marconi 100 supercomputer at CINECA (a not-for-profit Consortium, made up of italian universities and institutions).

In addition, a replica of the Marconi 100 supercomputer has been created in a virtual reality environment that allows the data from each node to be visualised concurrently. The ability to visualise in real time all this information, such as the power consumed, the temperatures of the cores (GPU and CPU) or the prediction of anomalies of the autoencoder model, will facilitate the task of detecting possible errors and will help the maintenance team.

In this paper we have two main purposes. The first one is to develop a robust anomaly detection system for supercomputers. We have to take into account that most of the supercomputer error data is not labelled. The second purpose is to create a virtual replica that allows us to visualise the data as a whole and in a simple way with an intuitive and highly usable interface. Furthermore, this should be possible without the need for technical knowledge of databases and command line.

To validate the proposal, the accuracy of the anomaly predictions as well as the usability and understanding of the virtual replica environment will be evaluated with different metrics. In addition, a set of modifications will be required, following the interaction guidelines of virtual reality devices, to run this system with virtual reality devices. Ideally, once this process is finished, the system should be tested by professionals working with HPC systems.

The rest of the paper is organized as follows. Section 2 reviews the state of the art of anomaly detection. Section 3 covers the implementation of autoencoder models and the creation of the supercomputer's virtual replica. The results are explained in Section 4. Finally, some conclusions are drawn in the last Section 5.

2 Anomaly detection: concepts and literature review

Anomaly detection is present in the industry as it saves time and money [6]. This was clearly demonstrated in a study carried out with Whirlpool microwave ovens [7]. In the paper, a real-time anomaly detection system using machine learning algorithms was created and evaluated, with good results.

In other works about anomaly detection, machine learning autoencoder have been used, with layers called Long Short-Term Memory (LSTM). This type of network arises as an improvement on recurrent neural networks, which have what is known as short-term memory. But as the sequence of data passes, the information vanishes and is of little importance for decision-making. This means that problems appear when you want to have a more long-term memory. It is at this point when LSTMs come up as a solution to the problem of long-term memory. LSTMs were first defined in 1997 [8] and are characterised by the fact that they are able to select the most important information to "remember" and preserve it for several instants of time. They can therefore have both short-term memory (like recurrent networks) and long-term memory.

On the other hand, the autoencoder networks [9] are a type of neural network that presents a bottleneck architecture from which the dimensionality of the input data can be reduced

while maintaining the information. The latent space is the low-dimensional representation of the data that the network learns. Therefore, this network is divided into two parts: an encoder that reduces the dimensionality of the input data and a decoder that is able to reconstruct the input data from the latent space information.

Autoencoder networks are widely used to deal with complex data that cannot be labelled in a simple way. Classification tasks can be solved from the latent space or by reconstruction error. The use of autoencoders has become a standard for solving some types of problems. One example of use is in image analysis tasks. Chen et al. [10] proposed a system that could extract information from medical images to detect anomalies and possible pathologies. By using autoencoders, they achieved good results and, in addition, unsupervised training is carried out, which is a great advance as all the images do not need to be labelled by expert healthcare personnel, which was very costly. Continuing in this field, Chicco et al. [11] used these networks for the annotation of genome information. This was a breakthrough as existing databases of known gene functions are incomplete and prone to errors. Through experiments with gene annotation data, it could be shown that deep autoencoder networks achieve better performance than other data reduction techniques, including the popular truncated singular value decomposition.

The use of these techniques is also very important in the field of IoT. For example, wireless sensor networks (WSN) are fundamental to the IoT by providing a bridge between the physical and cyber worlds. Anomaly detection is a critical task in this context, as it is responsible for identifying various events of interest, such as equipment failures and undiscovered phenomena. In the work of Singapore University of Technology and Design [12], autoencoder neural networks are introduced for the first time in WSNs to solve the anomaly detection problem and demonstrated through experiments to achieve high detection accuracy and low false alarm rate. Our work will focus on these lines of research and will be discussed in more detail in the next section.

Real-time monitoring of cloud resources is crucial for many tasks, such as performance analysis, workload management, capacity planning and fault detection. Progress has been made in monitoring and tracking systems in the cloud. The challenges of Big Data in industry and the complex working conditions of machines are raised and cloud-based solutions are proposed. To limit computational costs and ensure high reliability in capturing relevant load changes, an adaptive algorithm for monitoring Big Data applications that adapts the sampling intervals and frequency of updates to the characteristics of the data and the needs of the administrator, is presented by the work mentioned above.

We find cases in which the monitoring task becomes very difficult due to very high sampling frequencies and high computational cost, which leads to high economic and information management costs. A study has presented an adaptive algorithm for monitoring Big Data applications that adapts the sampling intervals and frequency of updates to the characteristics of the data and the needs of the given applications [13]. The adaptivity thus allows limiting computational and communication costs; and ensures high reliability in capturing relevant load changes. According to the researchers, this algorithm improves the state of the art.

This real-time adaptive control algorithm consists of two phases: a training phase for the evaluation of the best parameters for monitoring and an adaptive monitoring phase, the central part of the algorithm. In the first phase, the monitored data is analysed and a distinction is made between periods of relative stability and periods of high variability. The idea is to reduce the amount of monitored data when a resource is relatively stable, and increase it during the phase of high variability. In this way, the computational load is limited, and at the same time we ensure that important changes in the system are not lost. This algorithm works by dynamically fixing two variables: the sampling interval and the variability. The sampling interval determines the time that elapses from the collection of two consecutive samples. The smaller the sampling interval, the greater the number of data to be collected and transmitted. Variability represents the deviation between consecutive samples and is used to discriminate steady states from variables. When it is low, the monitored resource is considered to be in a stable state.

According to a study presented in [14], conventional detection approaches are based on statistical and time-invariant methods, which do not allow addressing the complex and dynamic nature of anomalies. With advances in artificial intelligence and the growing importance of anomaly detection and prevention in various domains, artificial neural network approaches are allowing the detection of more complex types of anomalies, taking into account temporal and contextual characteristics. Approaches using short-term memory appear to be the most successful.

Along these lines, the Computer Science and Engineering Department of the University of Bologna and CINECA have been researching for years to find solutions to anomaly detection in large distributed systems such as supercomputers or warehouse-scale computing. In their research work [15], they proposed the use of autoencoders to detect anomalies. In the training phase, both normal and anomalous examples must be provided to ensure the success of the learning task. The network must learn to correctly encode normal data and this allows anomalies to be detected since an abnormal data will not be encoded correctly and will produce a high reconstruction error. This problem manage Big data because to perform the task, data is handled using the Examon database in windows between 5 s and 10 s. In the case of this work they used the data from the nodes of the D.A.V.I.D.E.

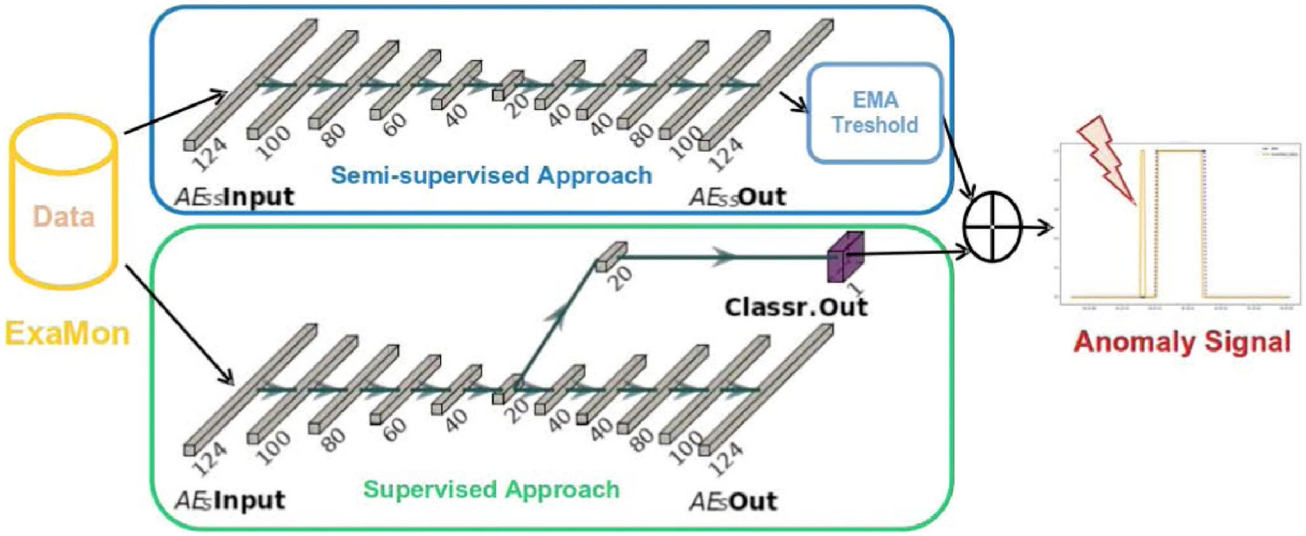


Fig. 1 Anomaly detection architecture used by the University of Bologna in the work [16]

(Development of an Added Value Infrastructure Designed in Europe) supercomputer, extracting around 170 features, i.e., the loads of the cores, temperatures, fan speed, power consumption of complete nodes, consumption of individual subcomponents, etc.

Once the autoencoder is trained, it can be used to process the data in real time and detect anomalies using the reconstruction error (input - output). This reconstruction error can be calculated in different ways. The reconstruction error is the difference between the output and the input. This difference is usually calculated per feature and then averaged (also called normalised). Alternatively, the maximum error can be used and in this work, interestingly, they chose that error, after a preliminary exploration. If the corresponding error is greater than a certain threshold, the data is classified as anomalous, normal otherwise. To decide the threshold they did a preliminary analysis in which they came to the conclusion that it was better to use a certain percentile of the error distribution of the normal data set and by means of the F-score determine the threshold.

They comment that the limitations of this system must be taken into account, as it clearly limits the type of anomalies that can be detected: events that last for periods shorter than the aggregation time window (5 min) and that do not leave any trace or permanent damage will not be taken into account. In addition, questions may arise, such as whether it is better to have a model for each node in the cluster, where each model is a particular neural network with its own hyperparameters, or rather to have a single model that applies to each node.

The same research group later came up with another broader study continuing with this [16] concept. The paper discusses the possibility of extracting tags from a service monitoring tool (Nagios), currently used by system administrators managing supercomputers, to mark nodes undergoing maintenance operations. This makes it possible to automatically annotate the data collected by a monitoring infrastructure and to use this tagged data to train and validate a model for anomaly detection. In this case they have performed the experimental evaluation on the level 0 production supercomputer, Marconi-100, hosted at CINECA, Italy.

From this progress we can highlight that a new architecture has been used, formed by two autoencoder neural networks, using one to calculate the reconstruction error and the other uses the latent layer (with the 430 input parameters) with a classifier to determine whether or not it is anomalous. By adding these two predictions, they say that the hit rate has been greatly improved and, in addition, the insurgence of anomalies can be predicted, with an average lead time calculated on historical traces of about 45 min. A diagram of the architecture used is shown in Fig. 1. This proposed technology [17] can be easily scaled up to other types of systems and thus be easily maintained. Regarding the use of a model for each independent node or a general one for all nodes, they point out that the difference obtained in hit rate has been minimal and although it is true that training a network exclusively with data from a single node improves the hits, the computational load is much greater than training a single network for all nodes. Therefore, they recommend using one model for each node only in critical cases where there is sufficient computational capacity.

3 Supercomputer virtual replica

Because of the number of components, the monitoring and maintaining of the status of supercomputers is not a trivial task. Along these lines, answers for this issue are being looked for, figuring out how to control the states and to

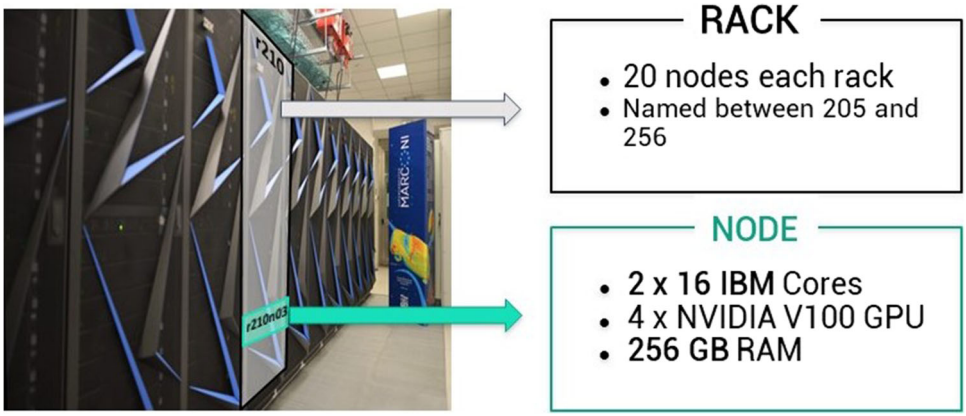


Fig. 2 Marconi 100 architecture

anticipate potential errors. Moreover, supercomputers do not have a similar architecture, despite the fact that they are frequently comparable, and a specific solution must be designed sometimes.

We are working with data from the nodes of the Marconi 100 supercomputer, provided directly by CINECA. We are dealing with thousands of features stored in very short periods of time, less than a minute. These data will have to be analysed and preprocessed before creating and training the anomaly prediction model.

3.1 Marconi 100 supercomputer

Marconi 100 is comprised of 55 racks. Each rack has 20 nodes inside it, as shown in the Fig. 2. Each node is composed of 2 x 16-core processors and 4 GPUs. As less relevant facts, with a performance of 32 Pflop/s, it was the ninth largest supercomputer in the world in 2020 and uses the Red Hat Linux distribution as its operating system.

The data used in our study is extracted from an ExamonDB database. This is a CINECA database in which all the supercomputer information is stored. Examples of this data are temperatures of both the CPU cores and the GPUs of the nodes, power, voltage and memory consumed by each CPU. And one of the most important data for us is the status, which indicates whether the node is working properly or has a problem. A value of 0 means that there are no problems and any other value means that there is a problem. We can get this kind of information thanks to the Nagios monitoring system that is integrated in the supercomputer, and from which all the data is logged.

Since it is a very large amount of data and we want to facilitate its management, it is stored in a different file for each node, that means 980 files. Apache Parquet file format has been used because of its benefits. It is an open-source, column-oriented data file format designed for providing efficient data compression and performance. Each file is named by the node name.

For each node we have 116 different metrics that are updated at different time intervals ranging from several seconds to minutes. Data from sensors, such as temperatures, voltages or memory, are stored in the Examon database. On the other hand, we have the log data provided by the Nagios monitoring system. These features have been processed in such a way that they have all been binned into time intervals of about 15 min. For each 15-minute interval, 4 features have been extracted for each characteristic: the minimum, the maximum, its mean and its variance. This allows us to reduce the amount of data while maintaining the variability and information produced by the data.

We must take into account the nature of the data and where it comes from. In our case, we know that it comes from the supercomputer and that it is sensor data at 15-minute intervals between July 2020 and April 2021. After preprocessing, it makes a total of 12,409 rows and 463 columns (features) for each node. A basic and very necessary analysis when dealing with many similar variables is the search for linear correlation between them, generating a correlation matrix. This results in a graph in the form of a heat map where values close to 1 indicate a positive correlation between the pair, an increase in the values of one variable is reflected in an increase in the other variable. In case of having values close to -1 we can say the opposite, when the value of a variable increases, the value of its analysed pair decreases. Finally, if the correlation value is close to 0 it means that there is no clear linear relationship between these variables.

Looking at the heat map in Fig. 3 we can see some relationship between the variables. There is a clear positive relationship between different system temperatures (both CPUs and GPUs), which is logical, given that when the computational load increases or decreases, it is distributed among all the cores, as they are usually parallelizable processes. We also observe that there is a correlation between the power supply and the GPU load, as these are power-hungry devices. The power supply is also correlated with the speed of the fans, as the more power the node consumes, the hotter it gets and the faster the fans should be. Finally, we can see how CPU load

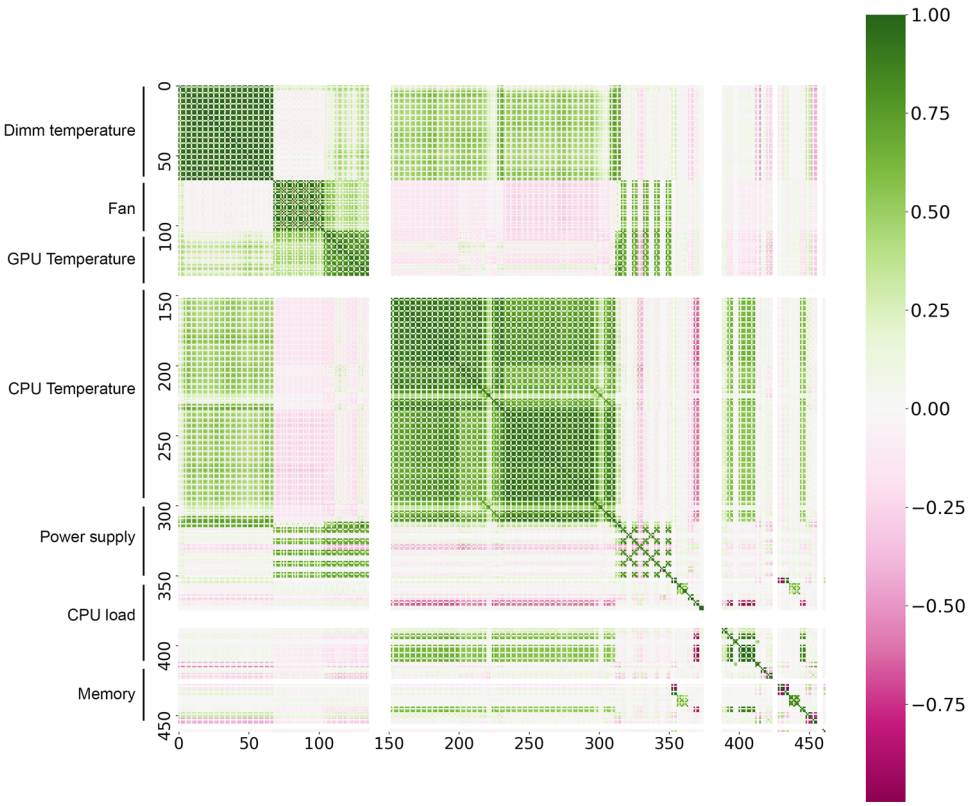


Fig. 3 Feature correlation heatmap of one node

is positively correlated with frequencies and temperatures, as these are controlled according to their needs.

After analysing the linear correlation between pairs of variables we can say that there are many variables that contain the same information. This can be clearly observed and is to be expected in variables representing DIMM temperatures or fan speed. The main reason why this happens is because each node is formed by many identical cores (with the same devices, fans, memory, computational capacity) and since most of the tasks performed in the supercomputer are parallelized in several processes, we have a homogeneous distribution of computational needs among the cores of the node. This causes the sensor readings of the cores to be very similar and therefore there is a high positive linear correlation. Moreover, in metrics such as temperatures and fan speed, it is easier to find this equality because they are variables that vary very slowly over time as well as with variations in the execution of CPU processes.

3.2 Anomaly prediction models

As we have seen before, supervised learning techniques are often used to train anomaly detection models. But this requires having a labelled and balanced dataset, as well as being unbiased. This is not an easy task since it is not always possible to generate labelled data from supercomputers and a need emerges to force the occurrence of errors in order to have more samples of anomalies. This approach is being carried out but it cannot be our case as we can only use our dataset.

Then, the possibility of using unsupervised learning comes up, which only requires the data with its characteristics but without labelling. During the training process the model must use the data to classify in a way that allows us to separate normal data from anomalous data. This is theoretically feasible but in practice does not give the expected results as there are many other ways in which data can be classified and can produce noise when classifying. Finally, there is another hybrid method known as semi-supervised learning, which uses labelled data to improve training but does not require all data to be labelled. Using this technique we can take advantage of the labels we have of normal and anomalous data to select only the part we are interested in, in our case the normal data. Next, a model can be trained with all the data to learn which are the normal values of the nodes. Using autoencoders, the reconstruction error of the data will be used to decide whether a sample belongs to normal or anomalous data. Therefore, it has been decided to take advantage of the latter approach to develop a system capable of detecting future errors in the operation of the supercomputer nodes.

An autoencoder can be viewed as a set of constraints that force the network to learn new ways of representing the data, other than simply copying the output. A typical

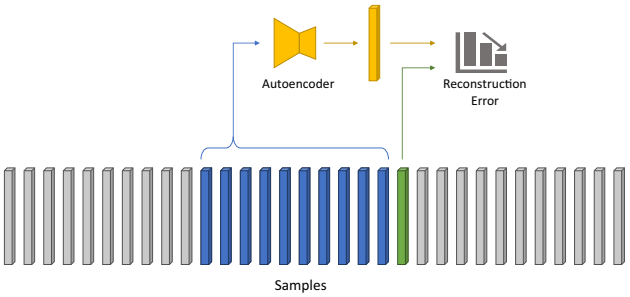


Fig. 4 Data flow and reconstruction error using autoencoder

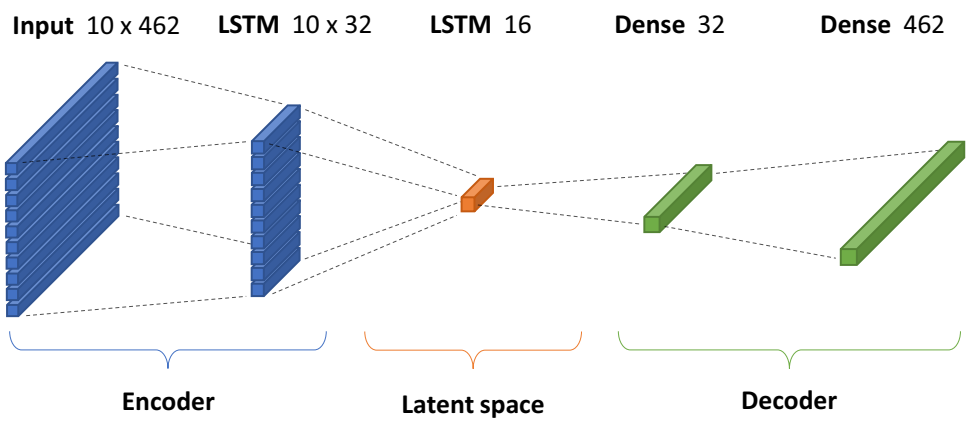
autoencoder is defined with an input, an internal representation (latent layer) and an output (an approximation of the input). Learning takes place in the layers linked to the internal representation. In fact, there are two main blocks of layers that resemble a traditional neural network, the encoder and the decoder. The slight difference is that the layer containing the output must be equal to the input. The idea is to pass the data as input and have it learn to reconstruct it from compressed data that retains maximum data variability. Passing as input only the node data at an instant in time would make it very difficult to detect an anomaly from the reconstruction error since anomalies tend to occur progressively over time, meaning that one of the most important factors to take into account is the difference between the data at one instant in time and the previous ones, not just the absolute value.

Therefore, a variation of the classical autoencoder is proposed in which instead of passing only the node data at one time instant, the input is the data from several consecutive time instants. The data collected from the old D.A.V.I.D.E. supercomputer [15] and the current Marconi 100 supercomputer suggest that symptoms of a possible error may appear up to two hours before it occurs. So considering that each sample is collected in a 15-minute time interval, it has been decided to introduce 10 samples as input, which represents a window of 2.5 h. In this way, the network can take into account data from a larger time region and detect patterns to predict the current time instant. The error between this prediction and the actual data, known as reconstruction error, will allow us to discriminate between normal and anomalous values, as shown in the Fig. 4.

To train this network correctly, we have to work exclusively with data in the form of time series, not being able to shuffle the data in any way to separate the training and validation set. This forces us to maintain the temperate order of the samples and to detect possible time jumps between data (when a node stops due to an error, we no longer have information about it). In fact, we have preprocessed the data looking for those temporary spaces without data, sectioning them and generating a list with dataframes of consecutive samples. This produces one more dimension but allows to generate inputs of blocks of 10 consecutive samples without errors.

Therefore, the data have been separated into three sets. One with the sequence to be used for training which will have the first 80% of the data. Another one with the next 10% that will be used as validation to take the best cut-off threshold for the reconstruction error of each model. And finally, the last 10% of the data set will be used to evaluate the best selected model with the validation data. Once the model and the parameters to be used have been defined, we move on to the training phase, which can be done in different ways. We will test the two approaches described in the University of Bologna article [15] on anomaly detection: a first one, in which we train a single model with all the data. And a second one that will train a model for each of the nodes. In this particular case we have to take into account that we are dealing with time series and forecasting events. This makes it even more difficult to evaluate the predictions. We know that an anomaly can be predicted minutes before it occurs, so we have to evaluate the predictions by analysing future samples in a given time window to know with more certainty whether an early anomaly prediction is a false positive or an anomaly detected well in advance. It is difficult to define a methodology to address this issue, but knowing that system failures have been predicted up to 50 min in advance [16]. It

Fig. 5 Autoencoder with LSTM encode layers



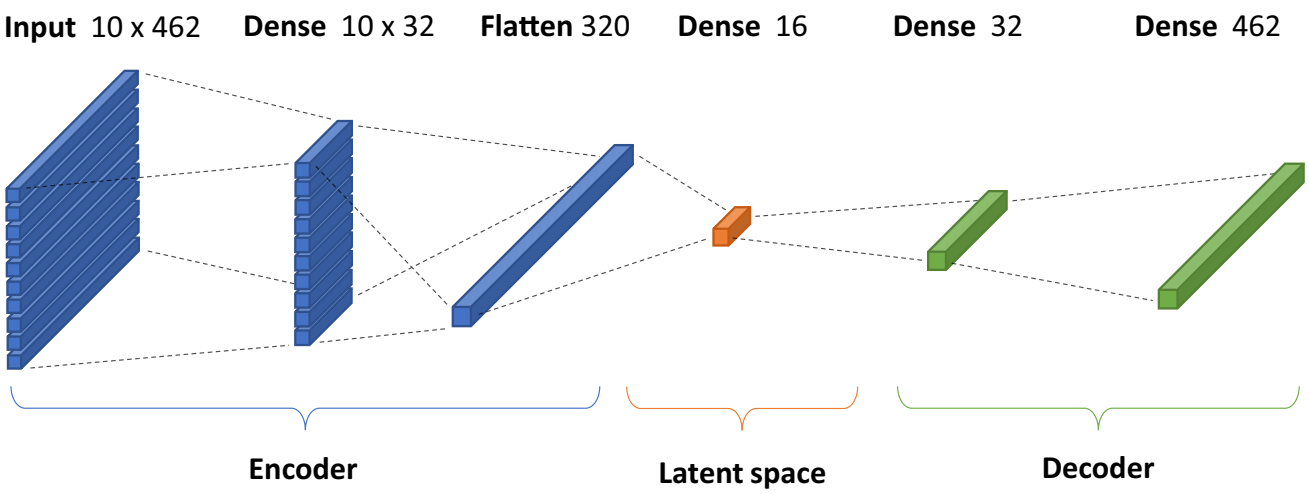


Fig. 6 Autoencoder with Dense encode layers

was decided to use a window of 45 min (3 samples after the prediction) to check whether or not there is an error detected and therefore the anomaly is a true positive.

In this case, we will train a single autoencoder with data from many nodes. Taking into account that each node has about 12,000 samples and that our computational and memory capacity is limited, it has been decided to use data from 75 randomly chosen nodes. This makes a total of almost 1 million samples, 80% of which were used for training and 20% of which were used for testing. This is enough data so that this is not a problem and we do not have to look for a more powerful machine to train the model. The encoder is made up of LSTM layers as shown in the Fig. 5.

As a second approach, we propose to use the same autoencoder but with dense layers instead of LSTM. Dense layers are those in which all the neurons are connected to all the neurons of the next layer. This type is more effective and generic in terms of information processing because each neuron has more data to process, but at the same time it takes more time in processing the data. Figure 6 represents this model with dense layers.

3.3 3D virtual replica

Data visualisation and the way in which data is displayed is a very important part of analysis and decision-making, especially when dealing with complex systems and many variables, which often have no trivial meaning. In this respect, a correct way of visualising data can greatly improve the task of working with this information. There are already cases where augmented reality applications are used to visualise data from industrial systems [18], making it easier for maintenance personnel to perform their tasks.

In this case, real data from a supercomputer has been used to create a virtual replica of it from which important data from all the nodes can be observed, something that would greatly facilitate the work of maintaining the system, given that the supercomputer nodes barely show any information through their physical interface (they only have a couple of lights that indicate the state of the current, data transfer). A virtual replica gives us the great advantage of being able to represent any type of data and graphics for each node, maintaining the 3D spatial structure of the nodes and racks of the system. This visual similarity between the real system and the virtual replica will greatly facilitate the task of reading data and contextualising it.

For this purpose, the visualisation and 3D graphics creation library VTK (Visualization Toolkit) for Python has been used. This library allows the import and visualisation of three-dimensional meshes, as well as the creation of 3D graphics from data and primitives (planes, cubes, texts). Therefore, a pipeline can be created that processes the data chosen from each core and processes them to display them in a 3D environment very similar to the real one.

The library is organised in such a way that each class implements a very specific functionality within the pipeline. In addition, everything can be structured in a few object categories. So the VTK visualisation pipeline consists of the following parts:

- Sources, used to read raw data.
- Filters can then be used to modify, transform and simplify that data.
- Mappers are responsible for creating tangible objects with the data.
- Actors, which are responsible for encapsulating these objects in a common interface that has many freely modifiable properties. These Actors are the ones that are passed to our renderer to show them in the scene.

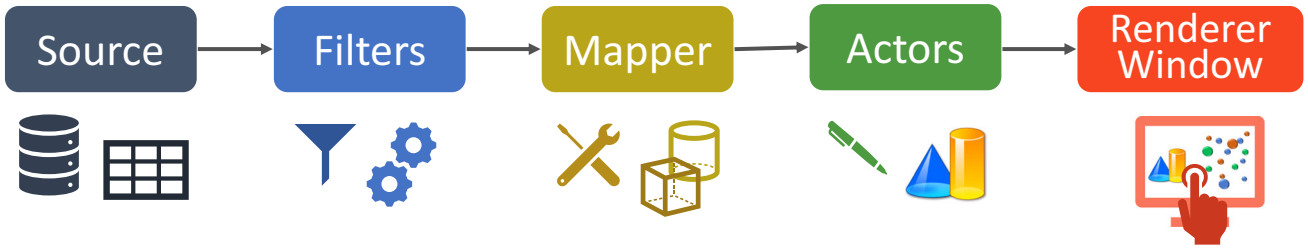


Fig. 7 VTK pipeline of the virtual replica

A diagram is shown in Fig. 7. In addition to all this, other much more specialised classes can be created that allow us to develop a more interactive environment, among other things. In this case, we used this to create keyboard shortcuts that we use to move around the time axis when displaying data.

The reading of data, used to create source objects, requires a lot of computing time, so it is interesting to parallelize the process. And in this case it is very easy to do so, as the data for each node is in a separate file and inside Python we store it in an individual table. So we have used an instance of *ProcessPoolExecutor* from Python's *concurrent.futures* module, which creates as many threads as nodes we have to read and handles them automatically depending on the capacity of our CPU. Doing this, it was managed to triple the loading speed, using my PC that only has 6 cores.

Creating the 3D model within VTK is easy thanks to the *vtkSTLReader* class, which allows us to import 3D files in STL format. In this way, Marconi 100 3D models provided by the CINECA visualisation team have been used. VTK library offers different classes that are responsible for carrying out different parts of the process. VTK objects has been created and the information has been injected from Pandas Data Frames. These source objects are in charge of transforming the data introduced as input (integers, floats, vectors) into a format understood by the rest of the objects in the VTK pipeline. Once this is done, we proceed to create the mappers. Each mapper is different for each type of representation we want to make, for example, there is a spatial mapper for text, heat maps, three-dimensional primitives or images. These mappers are given a reference to the source object with the data to be used, so if the source object is modified, the mapper will also modify the visualisation as it updates the input data. Finally, an actor has been created for each of these mappers. This actor encapsulates that visualisation or object and gives us the possibility to apply transformations to it in the environment. For example, we place the actor with the information of each node in the corresponding location taking into account the 3D model of the supercomputer. We also scale each one since, for example, we want to have larger texts (such as those indicating the name of the rack) and smaller ones.

An environment was created with the 3D model of the supercomputer at a scale of 1:1, with a square-shaped label to indicate the status of each node, using a colour (green if there are no errors, red if there is an error and grey if there is no data at that instant of time). To locate and position the elements in the three-dimensional space, it is necessary to know the location of each node of the supercomputer. A function has been created to calculate the position in space of a given node. This function uses the distances between racks and nodes, as well as taking into account the number of rows and the distribution of the racks in the room, returning the 3 coordinates of the particular node. It started to develop this first simple but functional version of the replica during my stay at CINECA.

An *actor2D* has also been created in which the day and time of the information being displayed is shown. An object of class *vtkTextMapper* is connected to this actor, which allows us to enter and modify the date in text format. In addition, all actors allow us to modify their size and colour. This functionality has been used to make the informative white text at the top of the screen, so that it stands out against the dark background. The 3D version of the *textMapper* has also been used to place three-dimensional labels on top of each rack indicating its name, as well as a text showing the consumption, in watts, of each of the nodes. The colour of this text varies according to its value. The next funcionality that has been added is the temperatures of the CPUs and GPUs. This visualisation is not trivial and we need to be very aware

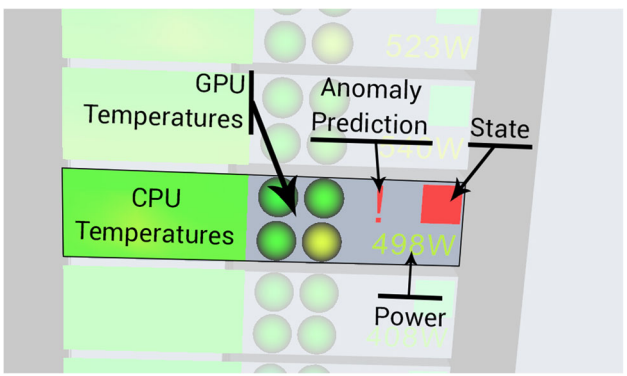


Fig. 8 Visualisation of the parts of one node

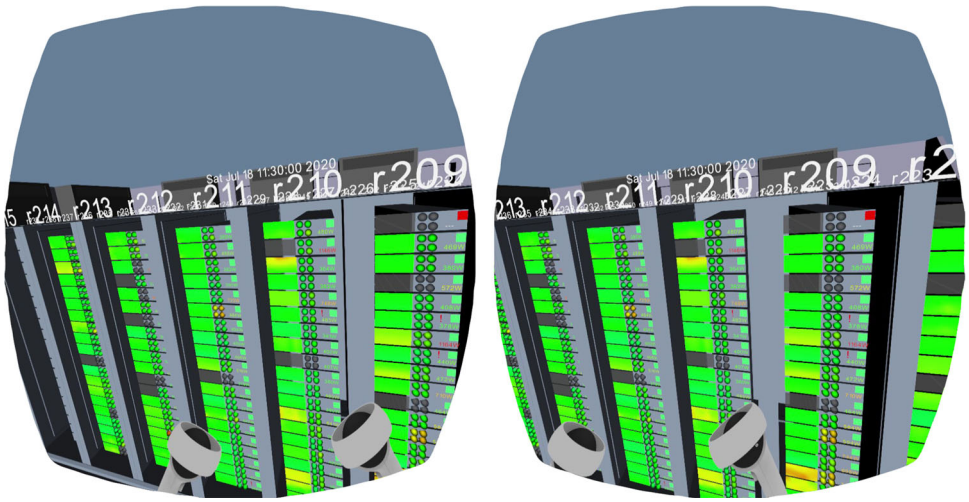


Fig. 9 VR view of the scene

of the nature of this data to be able to represent it. On the one hand, we have two processors with 16 cores each. This makes a total of 32 temperature values at each time instant since we have one value per core. On the other hand, at each node there are four graphics cards of which we have one temperature value per graphics card. Missing data is painted with a translucent grey colour.

As one of the purposes of this virtual replica is to be used as an industrial tool to detect possible failures in high performance systems, one of the most interesting things we can add to the visualisation is the real-time prediction of our anomaly detection models. For this reason, a red exclamation mark has been created using the *vtkVectorText* and *vtkTextMapper* classes. In the mapper object we have stored the polygonal information of the symbol. To optimise the loading of the scene, we will use the same object for each of the nodes. This is done by creating an actor (*vtkActor*) every time we need to put the symbol in a node, but always passing the reference of the same mapper as a parameter. This way, we save memory and computation as the scene is very cluttered with all kinds of objects in large quantities. Therefore, we have created a function that we call at each time instant asking for anomalies in the nodes. This function queries the predictions made for that time instant and if an anomaly has been predicted we will add the exclamation mark to the node in question. In our case, the function queries the predictions previously made because we do not have access to the data in real time and we have had to make a simulation of the performance. In a real case, we would call the prediction model with the data from that time period and the previous ones and wait for its prediction. Figure 8 shows the parts that make up a node.

3.3.1 Virtual reality integration

The VTK library enables the creation of virtual and augmented reality environments. The developers themselves explain in the paper [19] which are the guidelines to follow in order to avoid problems in development. It is also a library widely used by the scientific community to visualise virtual reality simulations [20].

In order to create a virtual reality environment with this library the OpenVR module is needed, which adds new Render VTK classes that inherit from the default ones and adds the connection with the OpenVR library. This module is disabled by default because it uses the OpenVR library which in turn can use other libraries such as the Oculus SDK, and this can cause problems. It is convenient to know that in order to run the virtual replica it is necessary to have SteamVR installed, which is the software that manages OpenVR. The easiest way to activate the VTK virtual reality module is to install the Para View program, which apart from bringing a Python interpreter installed with the Pandas, NumPy and VTK libraries, allows us to modify the last one, thus easily activating the modules we require.

Based on the version of the virtual replica seen above, it has been modified to allow it to run in a virtual reality environment. As we want to be able to visualise it both on a screen and with a virtual reality device, it has been developed with an architecture that allows to modify only the code related to the visualisation and interaction of the environment, keeping the logic and data representation part intact. We have a parameter that indicates whether we want to run the environment with the default render classes (OpenGL for screens) or the render classes of the OpenVR module (virtual reality). In addition to changing the render objects, there are a couple of things to change that are related to the interaction you have in VR. Now, to move around the scene, the jostick of the VR device controller can be used. Instead of the day and time label being displayed in two dimensions at the top of the screen, when you are in the virtual environment it is displayed as three-dimensional text on the ceiling of the room. These are little things that make the user experience greater when viewing in virtual reality.

The movement around the scene as implemented in VTK for OpenVR, you can move around in two different ways: with the jostick as mentioned before or by physically moving yourself around the environment, using the motion data captured by the headset to update your position within the virtual environment. If you have a large space it can be very useful to move around to check the nodes.

In Fig. 9 we can see a frame of the view of the environment using the virtual reality headset. The VTK library detects the controllers and shows them in the scene, this is very useful because in order to perform actions, it is sometimes necessary to know where the controller is located. In our case we only need it to navigate through the VTK menu. In this menu we are allowed to close and open the environment, and decide how we want the interaction to be: it can be a first person movement using the jostick (the default setting) or moving the 3D models by grabbing and dragging them (in third person). In this kind of environment, the first option is definitely the best. The second option is more suitable for environments with small objects that we can manipulate in a limited space.

Regarding the performance of the virtual reality environment, it consumes more resources than the desktop version. Having to render so many objects and make calculations such as heatmap gradients can really overload the CPU and GPU. In our case we have a relatively powerful desktop computer and even without displaying data from all the nodes it has a less than optimal performance. On average we get 30 FPS, when the recommended performance for any virtual reality system is at least 50 FPS. If we wanted to take this tool to a industrial use, we would need a very powerful computer or make some simplifications and improvements in terms of the visualisations of some data such as temperatures.

4 Results and discussions

Once the different models have been trained and tested on the same data, we can compare their results. We will test 3 different reconstruction errors: (1) the total reconstruction error, the sum of all the differences of the variables, (2) the maximum reconstruction error, maximum difference between the prediction of a variable and the actual value and (3) the mean squared error, which is the average of the squared differences of each variable. We use these errors to define a threshold from which to predict normal and outliers. To assess the performance of thresholds and errors we use the F-score metric. There are different ways to calculate the F-score, we can look at the F-score of a specific class, the average F-score of all classes or the weighted F-score, where classes with more data are more important. To establish this threshold, different values have been tested and evaluated with these metrics until the best one is found, testing error thresholds ranging from the 70th percentile of the normal data to the 100th percentile.

(a) Metrics of the model trained with data from all nodes, composed of LSTM layers

(b) Metrics of the model trained with data from all nodes, composed of Dense layers

Fig. 10 Comparison of the average F-score values obtained using total absolute error, maximum error and mean squared error as thresholds to make anomaly predictions. It was evaluated with test error thresholds ranging from the 70th percentile of the normal data to the 100th percentile

In Fig. 10 we can see the different average F-scores obtained when separating the data with different thresholds and reconstruction errors. We can see that we have very similar values for the different errors when using low percentile thresholds, below 90. On the other hand, the best F-scores are obtained when taking the 99th percentile as the threshold for the total error, and the 95th percentile approximately, for the maximum error and the mean square error. Is worth noting that the results obtained with the LSTM layer model and the dense layer model are very similar, so we cannot draw any conclusion about which of the two models is better with this metric alone.

Figure 11 shows the F-scores obtained using different errors as thresholds for the models trained on single-node data. We see how the mean square error, together with the total error, obtain on average higher F-scores. We can say

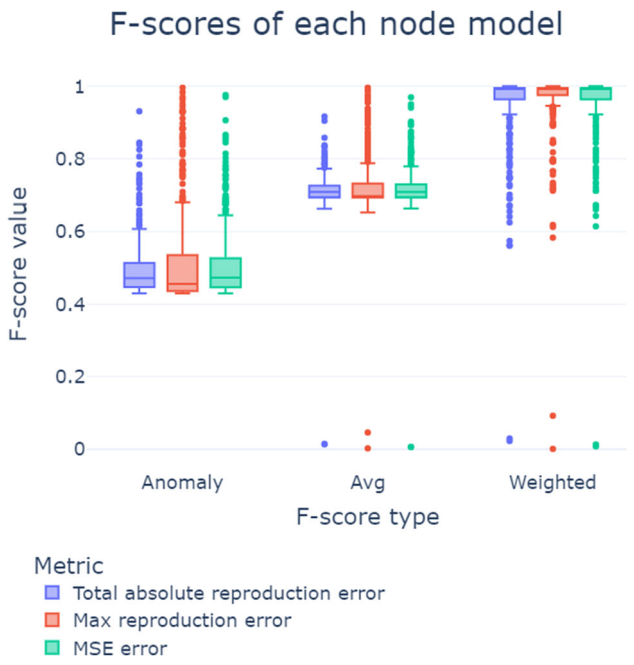


Fig. 11 Anomaly class F-score, average and weighted F-score box plot of all the node models with test data. Each point represents the score obtained by one node model

that, as detailed in previous studies, the use of one model per node gives better results. This means that there are differences in the anomalous behaviour of the nodes that single autoencoders for all nodes have not been able to interpret with the available data and this results in very low accuracy.

Moreover, comparing the performance of autoencoders with LSTM layer and dense layer encoders gives us an idea that the data at different time instants have a relationship that improves the prediction of anomalies. We realise this because the LSTM layer models have performed better when using their new features, such as feedback connections.

5 Conclusions

In this paper we proposed and tested some techniques to forecast anomalies in supercomputers by using deep learning techniques that permit a preventive maintenance. For this purpose, a sequence of data from the Marconi 100 supercomputer was analysed and processed, and different autoencoder models were trained to encode and decode the samples with the expected values from the supercomputer nodes. Among all the tests performed, the best result has been to train an autoencoder model with LSTM layers for each supercomputer node and to use the reconstruction MSE as a threshold to decide whether to classify the sample as normal or anomalous. After having entered and evaluated 240 models with their corresponding node data, the obtained weighted average F1 score is 0.98 for all the different reconstruction errors. These models can be used by supercomputer workers to perform preventive maintenance work, thus facilitating the detection of errors in the nodes.

In the second part of the paper, we have created a virtual replica of the supercomputer in which we can see the most important information of each node for an instant of time. This environment has been modelled as an industrial tool for professionals working on these high-performance systems. Among the features of the environment, it is worth mentioning that it allows the use of virtual reality devices, something that has not been integrated into CINECA's maintenance tools to date. This improvement will help to visualise the data remotely by simulating the real appearance of the supercomputer.

The parameterised creation of the entire pipeline makes it perfectly compatible to work with other high-performance systems. Only some modifications and improvements would be needed to make it fit the form and data of another type of system.

Acknowledgements We would like to thank "A way of making Europe" European Regional Development Fund (ERDF) and MCIN/AEI/10.13039/501100011033 for supporting this work under the MoDeaAS project (grant PID2019-104818RB-I00). Furthermore, we would like to thank the University of Skövde and to ASSAR Innovation Arena for their support to develop this work.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data Availability Not applicable.

Code Availability Not applicable.

Declarations

Ethics approval Authors agree with paper publication and consent to it. There are not ethical issues with the work.

Consent to publish The publisher has the permission of the authors to publish the given article.

Conflict of Interest The authors declare no competing interests.

References

1. Zamora-Hernández M-A, Castro-Vargas JA, Azorin-Lopez J, Garcia-Rodriguez J (2021) Deep learning-based visual control assistant for assembly in industry 40. Comput Ind 131:103485. https://doi.org/10.1016/j.compind.2021.103485
2. Villalba-Diez J, Schmidt D, Gevers R, Ordieres-Meré J, Buchwitz M, Wellbrock W (2019) Deep learning for industrial computer vision quality control in the printing industry 4.0. Sensors 19(18):3987
3. Angelopoulos A, Michailidis ET, Nomikos N, Trakadas P, Hatziefremidis A, Voliotis S, Zahariadis T (2020) Tackling faults in the industry 40 ERA-A survey of machine-learning solutions and key aspects. Sensors 20(1). https://doi.org/10.3390/s20010109
4. Sakellariou R, Buenabad-Chávez J, Kavakli E, Spais I, Tountopoulos V (2018) High performance computing and industry 4.0: Experiences from the disrupt project. pp 218–219. https://doi.org/10.1145/3229631.3264660
5. Hoe M, Dargham J (2020) High Performance Computing (HPC) applications in industry 4.0 (i4.0) for the betterment of humanity. In: 2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC). pp 1–6. https://doi.org/10.1109/R10-HTC49770.2020.9356990
6. Kamat P, Sugandhi R (2020) Anomaly detection for predictive maintenance in industry 4.0-a survey. In: E3S Web of Conferences, vol 170. EDP Sciences, p 02007
7. Stojanovic L, Dinic M, Stojanovic N, Stojadinovic A (2016) Big-data-driven anomaly detection in industry (4.0): An approach and a case study. In: 2016 IEEE International Conference on Big Data (Big Data). pp 1647–1652. https://doi.org/10.1109/BigData.2016.7840777
8. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9:1735–80. https://doi.org/10.1162/neco.1997.9.8.1735
9. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
10. Chen M, Shi X, Zhang Y, Wu D, Guizani M (2021) Deep feature learning for medical image analysis with convolutional autoencoder neural network. IEEE Trans Big Data 7(4):750–758. https://doi.org/10.1109/TBDATA.2017.2717439
11. Chicco D, Sadowski P, Baldi P (2014) Deep autoencoder neural networks for gene ontology annotation predictions. In: Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics. pp 533–540
12. Luo T, Nagarajan SG (2018) Distributed anomaly detection using autoencoder neural networks in WSN for IoT. In: 2018 IEEE International Conference on Communications (ICC). pp 1–6. https://doi.org/10.1109/ICC.2018.8422402
13. Andreolini M, Colajanni M, Pietri M, Tosi S (2015) Adaptive, scalable and reliable monitoring of big data on clouds. J Parallel Distrib Comput 79:67–79
14. Lindemann B, Maschler B, Sahlab N, Weyrich M (2021) A survey on anomaly detection for technical systems using LSTM networks. Comput Ind 131:103498. https://doi.org/10.1016/j.compind.2021.103498
15. Borghesi A, Bartolini A, Lombardi M, Milano M, Benini L (2019) A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems. Eng Appl Artif Intell 85:634–644. https://doi.org/10.1016/j.engappai.2019.07.008
16. Borghesi A, Molan M, Milano M, Bartolini A (2022) Anomaly detection and anticipation in high performance computing systems. IEEE Trans Parallel Distrib Syst 33(4):739–750. https://doi.org/10.1109/TPDS.2021.3082802
17. Borghesi A, Bartolini A, Lombardi M, Milano M, Benini L (2019) Anomaly detection using autoencoders in high performance computing systems. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 33. pp 9428–9433
18. Subakti H, Jiang J-R (2018) Indoor augmented reality using deep learning for industry 4.0 smart factories. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol 02. pp 63–68. https://doi.org/10.1109/COMPSAC.2018.10204
19. O'Leary P, Jhaveri S, Chaudhary A, Sherman W, Martin K, Lonie D, Whiting E, Money J, McKenzie S (2017) Enhancements to VTK enabling scientific visualization in immersive environments. In: 2017 IEEE Virtual Reality (VR). IEEE, pp 186–194
20. Kok AJ, Van Liere R (2007) A multimodal virtual reality interface for 3D interaction with VTK. Knowl Inf Syst 13(2):197-219