LSTM-Based Unsupervised Anomaly Detection in High-Performance Computing: A Federated Learning Approach

LSTM-Based Unsupervised Anomaly Detection in High-Performance Computing: A Federated Learning Approach

Emmen Farooq
DISI, University of Bologna,
Bologna, Italy
emmen.farooq3@unibo.it

Andrea Borghesi
DISI, University of Bologna,
Bologna, Italy
andrea.borghesi3@unibo.it

***Abstract*—High-Performance Computing (HPC) systems are intricate machines that must be run at maximum efficiency to justify their high cost and to minimize environmental impact. Any anomalies that hinder the smooth operation of supercomputing nodes are a significant issue in modern HPC systems. Therefore, the development of automated anomaly detection methods is a crucial area of research within the HPC domain. Machine Learning (ML) models have shown great success in identifying anomalies on individual nodes, especially as contemporary supercomputers are outfitted with advanced monitoring systems that provide large datasets for training. However, the potential to combine data from various nodes and to utilize collective ML models remains largely unexplored. Federated Learning (FL) presents a promising approach by enabling individual models to share and learn from one another. Although FL has been employed in areas like healthcare and IoT, its application in HPC is still novel. This study explores how FL can be leveraged to enhance anomaly detection in HPC systems. Using data from a real-world supercomputer, the approach has shown significant promise, boosting the average F1-score from 0.307 to 0.815, and the average AUC from 0.368 to 0.77. Moreover, FL drastically reduces the time required to gather sufficient data for training, allowing faster deployment of detection models. Traditional ML models typically need about 4.5 months of data to perform effectively, but FL can achieve the same with only 1.2 weeks of data, resulting in a 15-fold reduction in data requirements.**

***Index Terms*—Federated Learning, Long Short-Term Memory, LSTM, Anomaly Detection, High-Performance Computing, HPC, Unsupervised Learning, Machine Learning**

I. Introduction

High-performance computing (HPC) systems and warehouse-scale computing are essential aspects of modern society and industry. Contemporary supercomputing centres are sophisticated industrial infrastructures [1], usually organized in several computing rooms, each with multiple racks totalling hundreds of computing nodes [2]. The computing nodes typically comprise heterogeneous processing units like CPUs, GPUs, NPUs, and FPGAs. The jobs submitted by users are executed as parallel applications on the nodes. The demand for supercomputers has grown globally[1] and is expected to increase [3], leading to HPC systems being ever more complex and with more components.

The large scale of current and future HPC systems makes optimally managing them very complicated. A wide range of challenges needs to be addressed by system administrators and facility owners, ranging from investment costs and mortgage, power and energy consumption, system availability and maintenance. Detection of fault conditions is a very significant issue in the functional efficiency of supercomputers [4]. Anomalies can occur from various sources like incorrectly configured hardware (HW) and nodes, problems in the applications that run on HPC systems, job scheduler issues, and problems in the computing units [5].

HPC administration practices still strongly rely on the domain experience and the skills of the system administrators and system operators, who are tasked with analysing the huge amount of data produced continuously by the set of logging tools and data-collecting sensors that are typically deployed in today's supercomputers. In recent years, several research avenues have been opened to make sense of the huge amount of available data, especially relying on data-driven approaches from the Machine and Deep Learning (ML and DL) areas, especially from the predictive maintenance area [6]. Systems logs have been examined to analyze the cause of anomalies after they have occurred [7]. Several researchers have attempted to deduce meaningful information from large volumes of data collected in modern HPC systems [8], which have several sensors to inspect the status of their different components. The crucial need is to derive meaningful insights from such a large volume of data. Unsupervised or partially supervised DL approaches can be adopted when the data is not annotated (for instance, when there is no information about the status of a machine associated with the data sample describing the system at a given time), as demonstrated by several works [9]–[11]. When information about the status of the computing nodes and the HPC system is available, that is when the collected data is *labelled*, then supervised approaches are possible [12], [13]. This annotation step is enabled by

This work has been partially supported by European Project HORIZON-CL4-2021-HUMAN-01-AI4Europe (g.a. 101070000). The authors thank Huawei for providing the hardware used in this research and for their support.

[1]https://www.top500.org/

event monitoring and node status reporting tools often installed in modern supercomputers to assist system administrators by warning them of the occurrence of anomalies in nodes; the alerts generated about anomalies and faults are then verified by manual inspection by system administrators [14].

In this paper, we propose to apply Federated Learning (FL) to enhance anomaly detection performance of anomaly detection DL models for HPC. FL expedites the decentralized improvement and learning of independent DL models [15]. Although FL has been used to improve anomaly detection in other domains [16], it has only been partially explored in the HPC setting [17]. The potential of FL in this context lies in the fact that HPC nodes tend to fail independently from one another, but their behaviour follows similar patterns (at least, within the same HPC system).

In this paper we focus on unsupervised anomaly detection for HPC systems, as labelled data is not required; data annotation is very costly but unlabelled data is abundant in HPC systems, thanks to the widespread adoption of monitoring infrastructure. We use Long Short-Term Memory (LSTM) autoencoders to implement unsupervised anomaly detection as LSTMs incorporate temporal patterns and long-term dependencies in the data more accurately than regular Recurrent Neural Networks RNNs [18]. Using LSTM neural networks to implement anomaly detection in HPC has been recommended in recent literature [11], but we are the first to propose an FL-based enhancement. We demonstrate the effectiveness of our approach by validating it on data from a real, production HPC system, Marconi100[2] hosted at CINECA, the Italian supercomputing centre in Bologna. The main contributions of this paper are the following:

- We apply FL to improve unsupervised anomaly detection using LSTM autoencoders. The application of FL improves the average F1 score from 0.307 to 0.81.
- The application of FL also improves the average Area-Under-the-Curve (AUC) from 0.36 to 0.77.
- We demonstrate the adoption of FL results in a reduced amount of training data required to obtain accuracy comparable to the state-of-the-art. The baseline approach (no FL) requires 5 months of training data to achieve effective results in anomaly detection, while the FL method reduces the training data duration requirement to 1.25 weeks. The training data requirement is decreased by 15 times.

TABLE I: Average Number of Normal, Anomalous, Total Data Points and Average Number of Anomalies in Data Set 1 (D1) and Data Set 2 (D2)

Data Set	#Normal	#Anomalous	Total	% Anom
D1	12650.25	391.25	13041.49	3.00
D2	14200.75	241.41	14442.16	1.7

[2]https://www.hpc.cineca.it/hardware/marconi100

II. Related Works

Anomaly detection has been an important aspect in several fields like credit card fraud detection [19], energy sector [20], information technology [21] and the Internet of Things (IoT) [7]. In this work we focus on anomaly detection in HPC systems and on FL for anomaly detection; we discuss the latter and the former topics in the following subsections.

A. Federated Learning in Anomaly Detection

Anomaly detection has been facilitated by FL in several fields. The application of FL allows models to learn from each other and exhibit performance comparable to the central model.

FL has also been used to enhance the performance of long short-term memory models (LSTM) for anomaly detection in several fields. A federated framework, proposed in [22], trains on time series data generated from several institutions. The implementation of FL enhanced the accuracy, f-score, and recall of the LSTM autoencoder. In another research work, FL has been applied to enhance the performance of LSTM anomaly detection models for IoT sensors in smart buildings [23]. The application of FL made the anomaly detection models converge twice as fast as the central LSTM. The framework was tested on three real-world datasets at General Electric Current smart building and it reduced the overall training cost without affecting the anomaly detection performance. A communication-efficient on-device federated learning-based anomaly detection framework has been proposed in [24], for sensing time-series information in industrial IoT. FL is applied to allow decentralized edge nodes to collaboratively train an anomaly detection model. Then an attention mechanism-based convolutional neural-network LSTM model is proposed for anomaly detection. The proposed framework was tested on four real-world data sets for anomaly detection, and it reduced the communication overhead by 50%. A novel Stacked Convolutional Neural Network and Bidirectional Long Short Term Memory (SCNN-Bi-LSTM) model for intrusion detection in Wireless Sensor Networks has been proposed in [14]. The proposed model applies FL making several nodes train collaboratively. The application of FL enhances intrusion detection performance and gives 99.9% accuracy. In another research work [16], an FL approach has been proposed with an ensemble to facilitate anomaly detection in IoT networks. Long short-term memory (LSTM) and gated recurrent units (GRUs) neural network models have been trained on Modbus network information. The application of this research approach demonstrated a reduced error rate in attack prediction, and reduced frequency of false alarms as compared to the ML approach.

B. Anomaly Detection in HPC Systems

Anomaly are faults of HPC systems that cause failed or incomplete jobs, and failures of nodes and hardware [12]. Although there are several methodologies to reduce anomalies, their occurrence still reduces the computing time for which HPC machines can be used by users [25]. The progress

towards Exascale[3] computing capabilities and the increasing complexity of HPC hardware have made anomalies an even more critical issue [1].

Node specific anomalies in HPC, are still very rare events and can be considered a category of supervised learning on unbalanced classes [26]. The normal operation data of HPC systems is available in a larger volume than anomalous behavior data, thus due to this imbalance, supervised anomaly detection techniques are not easily applicable [9]. Data in which every point is labeled either as an anomaly or as a non-anomalous point is also not readily available in the HPC domain, thus this is also one of the reasons for which supervised anomaly detection approaches become unfeasible [27]. Unsupervised anomaly detection techniques, however, do not require any labeled data, and can thus be more readily applied to the HPC context.

LSTM autoencoder networks have proven to exhibit improved anomaly detection results as compared to regular autoencoders [20]. In the HPC domain, LSTM autoencoders have been used to implement anomaly detection [28]–[30] and have shown improved performance as compared to state-of-art methodologies, and as compared to anomaly detection methods using regular autoencoders [11]. This improvement is due to the time-series nature of the data collected on HPC and the fact that LSTMs naturally handle the intrinsic temporal dimension, contrarily to dense autoencoders that possess a more combinatorial nature.

In a previous work [17], we proposed the first application of FL to improve anomaly detection in HPC using regular dense autoencoders. Our current research is the first application of FL to improve anomaly detection in HPC using LSTM autoencoders; LSTM networks have a very different structure than regular, densely connected neural networks. Importantly, LSTMs take into account the temporal information encoded in the HPC data, which comprises multi-variate time series. For this radical change in network structure and for the explicit treatment of the temporal aspect, applying FL to this task is very different from our previous work - namely, any eventual improvement cannot be taken for granted, hence this exploration is warranted and worthy.

III. Methodology

The motivation of our research is to enhance anomaly detection in HPC nodes by using FL. We emphasize time series base unsupervised anomaly detection, which has exhibited exemplary results when applied to a real HPC system. It also eliminates the overhead of requiring labeled data for training. We use the same LSTM autoencoder neural network, that incorporates time dependency, implemented in [11] for time series-based anomaly detection in HPC. This work will be our *baseline* upon which we claim that FL can bring benefits. Earlier research elaborated on how to train an autoencoder for every HPC node separately and then use a threshold-based method to identify anomalies during prediction; this is what is generally done by the majority of works in the literature [9], [10], [13], [27]. Alternatively, the anomaly detection models are trained in a collaborative and federated manner, facilitating models to learn from each other and to improve performance. In this paper, we propose this second approach to train LSTM autoencoders for anomaly detection in HPC nodes. The workflow of our research is depicted in Fig.1. The core of the approach is to treat the different computing nodes as if they were client in a standard FL setting. In this way, each node has its own local model (similar to the baseline); then, the FL strategy allows for improving the local models by sharing information through the global model (exact details will be discussed later). We first elaborate on time-series-based unsupervised anomaly detection and then explain the FL architecture in the upcoming sections.

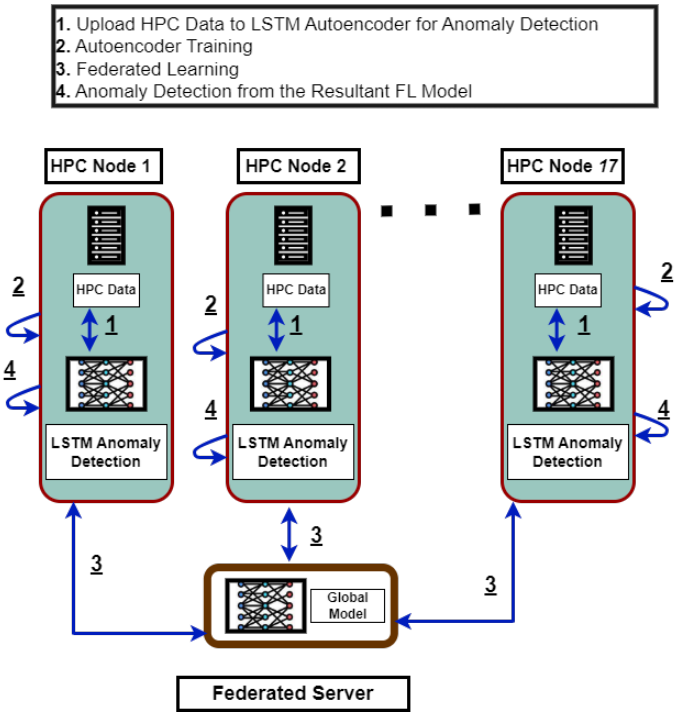


Fig. 1: Unsupervised Anomaly Detection with LSTM Autoencoders for HPC using FL

A. Unsupervised Anomaly Detection - Baseline Model

As the baseline model to detect anomalies, we employed an LSTM autoencoder neural network. The details of this network will be provided in Sec.IV-C. The LSTM autoencoder DNN was trained on unlabelled data containing both normal and anomalous data points. Then reconstruction error of the autoencoder, for new data points can be used to analyze if they are an anomaly or a normal point [14]. The reconstruction error of an autoencoder can be computed in different ways; a common method is to compute an aggregate over all the features, such as mean, min, max and other statistical aggregations – in this work, we opted for computing the mean error

[3]"Exa" means 18 zeros. That means an exascale computer can perform more than 1,000,000,000,000,000,000 FLOPS, or 1 exaFLOPS.

over all features in the data, drawing inspiration from other works from the literature [31]. If the autoencoder reconstruction error of a new data point is lower than a threshold, it is declared a normal point, otherwise, it is declared an anomaly. The value of the threshold is a vital aspect of the efficiency of the anomaly detection approach via autoencoders; see for instance a lengthy discussion in [9]. However, evaluating the best method to select a threshold is not within the scope of the current work. For this reason, the threshold was chosen by empirical exploration and by taking guidance from modern literature [11], [12]. We used the same value of threshold in non-FL and FL experiments for a fair comparison (which is the actual focus of this paper).

B. Federated Learning Enhancement

FL is incorporated because although the autoencoders explained in the last section are trained independently, each model can benefit from collaborative learning [32], as nodes present similar behaviour. Fig.1 depicts the architecture of the FL strategy we developed. There is a central server, which can be envisioned as the management node (or nodes) present in most standard supercomputers [33]. The LSTM autoencoders are the local models. The computing nodes act as the clients that communicate with the management server. There is also a daemon running in each computing node to gather node-related information with minimum overhead, which produces the data used in anomaly detection models [34]. There is an autoencoder model for each client which has been trained on its local data.

The server sends weights to clients to start the FL iterations. After completion of local training, each client sends its weights to the central server. The global model is updated by the server by aggregating all weights received from the clients. FedAvg algorithm [35] is used by the server. The weights obtained from all clients are averaged in FedAvg according to Eq.1.

$$\omega_{t+1} \leftarrow \sum_{k=1}^{k} \frac{n_k}{n} \omega_{t+1}^{k} \tag{1}$$

In the above equation, ω_{t+1} depicts the most recent weights sent to clients by the server. ω_{t+1}^{k} shows the weights sent by client *k* to server in the *t+1* round of communication. *t+1* is the current round of communication. *n* depicts the total number of data values used for training the global model, whereas n_k is the number of data samples used by client *k* for local training [4]. We opted for a basic FL strategy because of its proven success in similar contexts [24] and because we want to avoid complicating the experimental evaluation. All client autoencoders have the updated global model at the end of FL. Each of them executed unsupervised anomaly detection, again on their HPC data.

IV. Experimental Evaluation

In this section, we first elaborate on the two datasets used for our experiments. Then we explain the technical architecture of the anomaly detection model setup (the LSTM autoencoder). Finally, we discuss the implementation of FL. Our research experiments were executed on a system with 42 Intel(R) Xeon(R) Gold 6240R CPUs with 96 cores and 790 GB RAM. The system has Ubuntu 22.0 installed in it.

We focus our experiments on detecting anomalies happening at the level of compute node – we do not consider systemic anomalies or issues caused by the workload submitted by users. For this reason, we only consider measurements coming from the hardware sensors available in the monitoring infrastructure, such as the computing units' power consumption, the core temperature, the clock speed, and so on; we disregard workload information. Both datasets described below are composed by this hardware-level information that we use to characterize the behaviour of HPC nodes and by information describing the status of the node - as internally used by system administrators.

We want to stress out that both of the datasets that we employ come from a real, Tier-0 production supercomputer. The anomalies that we will detect are not synthetic or injected by us, but are the actual anomalies and faulty situations registered and tackled by system administrators.

A. Data Set 1 (D1)

The dataset used in this research has come from a tier-0 supercomputer, Marconi100[4], hosted at CINECA Bologna. It currently has 980 nodes. Each node has 2 CPUs composed of 16 cores each, 256GB of Random Access Memory (RAM), and 4 Nvidia Volta V100 GPUs. The cores of every node have IBM POWER9 AC922 at 3.1 GHz. The highest performance exhibited is 32 PFlops/s. Marconi100 is endowed with a holistic monitoring infrastructure that collects a wide set of various metrics that can be employed to build a faithful representation of the entire supercomputer. This monitoring infrastructure is called Examon and it has been used to collect data from a variety of HPC systems hosted at CINECA [34], [36], [37]; we focused on Marconi100 due to its relevancy as a Tier-0 machine. There are various measurements in the gathered data, such as information from sensors like fan speed, clock frequency, power utilization of CPUs and GPUs, information about system availability, job scheduler, and status updates. 462 metrics are collected for each node.

A software Nagios[5] has been used to annotate the dataset for our specific anomaly detection task. Nagios is used by system administrators to label anomalous nodes after users have identified them to be anomalous. The node status interpretation information provided by Nagios indicates whether a node is anomalous or healthy. Nagios labels nodes every 15 minutes, hence a 15-minute aggregation time duration is used as the sampling rate for the learning task (anomaly detection) faced in this work. This sampling rate has been demonstrated to be suited for the tasks in several previous works [9], [11], [13]. The temporal behavior of collected data is incorporated by determining the standard deviation and average value over the 15-minute duration. The target of anomaly detection i.e. the

[4]https://www.hpc.cineca.it/hardware/marconi100

[5]https://www.nagios.org/

label is either 0 showing an anomaly or 1, showing a healthy state of the node.

B. Data Set 2 (D2)

The second data set is of the same underlying machine (Marconi100) having the same nodes but the monitoring infrastructure was configured differently in its collection. In practice, the Examon monitoring infrastructure was significantly changed to expand its capabilities and to provide a more detailed snapshot of the operative life of Marconi100. This expanded view has been made publicly available for researchers and practitioners [38]; this public information is the source of the data used for our experiments.

The infrastructure used for collecting D2 was enhanced with software probes and physical sensors to improve its accuracy in characterizing the system's behaviour. There are 573 features collected for every node in this dataset. The major change compared to D1 is the labeling mechanism used to declare data points as anomalies, as it was refined by system administrators to concentrate on relevant anomalies, as indicated by a reduction in the average percentage of anomalies from 3% in D1 to 1.7% in D2 (Table I). Some anomalies declared as such by the labeling mechanism employed in D1 were recognized to be transitory conditions due to changes in configurations of the computing nodes and the operative system installed on them; in practice, it was decided not to consider them anymore as faulty conditions and hence were labelled as normal points in the collection of D2. This makes the adoption of D2 a different experimental use case, as the dataset contains different types of anomalies and a different set of measured metrics (with partial overlaps with D1); more importantly, there is *no temporal overlap* between D1 and D2 – the target system is the same (Marconi100) but monitored at different phases of its lifecycle, and with different tools. With these considerations, we can confidently claim that D2 is a significantly different task on which to evaluate the application of FL.

As our research work is the first evaluation of FL on time-series-based unsupervised anomaly detection on HPC, we decided to validate on 5 groups of 17 nodes each from D1 and D2. The same nodes are used in the AE baseline and FL experiments. We state that this number of nodes is reasonably high to ensure that our results are not random.

C. Implementation

TABLE II: Hyperparameters of LSTM AE

Name of Hyperparameter	Value
Time Steps	10
Optimizer	RMSprop
Learning Rate	0.000001
Loss	Mean Squared Error
Batch Size	128
Epochs	5
Reconstruction Error Threshold	0.65

We started with a pre-processing phase to prepare the data for the learning task. First, we removed the workload information and all non-numerical data. In particular, the status of the nodes obtained through Nagios is not going to be used to train the DL models. However, we do not discard this information entirely but we rather keep it separated from the DL models at training time; we use these labels instead at testing time to evaluate the prediction of the anomaly detection models. Then, "Null" and incomplete data points (i.e., data points that miss one or multiple features) were dropped. The remaining data was normalized in the 0 to 1 range. For this pre-processing operation, we used the scikit-learn Python library[6]. The data of the different nodes was split – each node is used to train a different detection model: The data of each node was split into 80% training and 20% for testing. LSTM autoencoders require input data to be converted in the shape of an array as $\{$*samples, TIMESTEPS, features*$\}$. A Time Step of 10 was used, as advised in contemporary research work [11]. As the aggregation window used in both datasets is 15 minutes, every autoencoder had a memory of $15 \times 10 = 150$ minutes. The architecture of the LSTM autoencoder is described in Fig.2. Table II reports hyperparameters used in time series-based unsupervised anomaly detection. The architecture of the LSTM autoencoder and hyperparameters of anomaly detection were determined after a manual, empirical exploration; the hyperparameter space was reduced taking inspiration from current state-of-the-art [**?**], [6], [13]. Every autoencoder was trained in an unsupervised manner. To detect anomalies we employ the reconstruction error of the LSTM autoencoder; the reconstruction error is computed as the mean over all features. The reconstruction error is compared to a threshold θ; if the error is smaller than or equal to θ the point is classified as normal, otherwise, it is considered anomalous. We used a threshold of 0.65, again determined after extensive empirical exploration and state-of-the-art considerations (see in particular [9]).

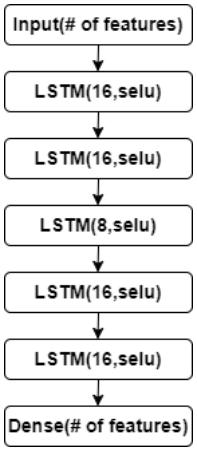


Fig. 2: Architecture of LSTM Autoencoder

Each autoencoder was implemented as a TensorFlow Keras[7]

[6]https://scikit-learn.org/stable/

[7]https://www.tensorflow.org/api_docs/python/tf/keras

model, and was executed as a Python script. Each node has an individual autoencoder, and batch scripts and Python programs were used to automate inter-node training. The performance of models was evaluated using standard anomaly detection metrics [14] namely Area-Under-the-Curve (AUC) and the F1-score.

Flower[8] framework has been used to implement the FL strategy. In the FL paradigm, clients and the central server need to be identified. In our setting, each HPC node represents a client (each one with its own local DL model - the LSTM autoencoders) while the server is the same management node that is used to perform maintenance on the HPC system and where the monitoring infrastructure is deployed as well.

We start by building a baseline LSTM autoencoder (as described above) for each node. These baseline models of each node are trained and evaluated *without any FL scheme*. Then we apply the FL strategy and train the model accordingly; we employ FedAvg [17] as the FL strategy. Each computing node participates in the FL interactions as a client (employing the FlowerClient implementation). In the literature it has been suggested that more than one thousand FL rounds should be employed [17]; we thus use 1000, 1500, and 2000 rounds of FL for our experiments. However, we must note that increasing the number of rounds did not bring any experimental advantage in our setting. For this reason, we will not explore this aspect further in the remaining sections.

D. Metrics

The following metrics were used for the experimental evaluation (following state-of-the-art practices [11], [12]): i) F-score and ii) Area Under the (ROC[9]) Curve, also called AUC.

We define the F-score as reported in Equation2.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2}$$

The AUC [39] measures the two-dimensional area under the ROC curve, with coordinates in the Cartesian plane from (0,0) to (1,1). It provides an aggregate measure of performance across all classification thresholds. It depicts the probability that a model ranks a positive example more highly than a random negative example.

E. Results

In this section, we state results that validate the efficiency of our research. We explore the following research questions.

1) **RQ1-** Does the application of FL to unsupervised anomaly detection in HPC improve the anomaly detection performance of models?
2) **RQ2-** Can the application of FL decrease the amount of training data required to achieve the same performance as non-FL, baseline anomaly detection models on full training data?

[8]https://flower.dev/

[9]Receiver Operating Characteristic

TABLE III: Experimental Results Obtained Over 5 Groups of 17 Nodes Each of D1 & D2

	F1-Baseline	**F1-FL**	**AUC-Baseline**	**AUC-FL**
	Data Set 1			
Group A	0.31	**0.86**	0.38	**0.74**
Group B	0.25	**0.78**	0.44	**0.77**
Group C	0.37	**0.87**	0.47	**0.85**
Group D	0.22	**0.79**	0.33	**0.69**
Group E	0.24	**0.84**	0.25	**0.63**
	Data Set 2			
Group A	0.51	**0.94**	0.54	**0.96**
Group B	0.42	**0.82**	0.42	**0.85**
Group C	0.17	**0.67**	0.22	**0.70**
Group D	0.3	**0.8**	0.33	**0.77**
Group E	0.28	**0.78**	0.30	**0.81**

The first objective is to investigate the potential of **RQ1**. Table III, depicts the results of the application of FL to improve unsupervised anomaly detection. The first five rows show results of the application of FL to improve unsupervised anomaly detection of 5 groups having 17 nodes each from D1. The last 5 rows of Table III, show the results of the application of FL to improve unsupervised anomaly detection to 5 groups from D2 having 17 nodes each; the groups are all composed of different nodes. The 1^{st} column shows the average baseline F1 of each group, meanwhile, the 3^{rd} column depicts the average baseline AUC of each group. The 2^{nd} column shows the average F1 of each group, and the 4^{th} column shows the average AUC of each group after the application of FL. The 2^{nd} and 4^{th} column of Table III, show that the average F1 and the average AUC of each group improves after the application of FL, hence authenticating **RQ1**.

1) Individual Node Analysis: In individual node analysis, we analyze the improvement in anomaly detection performance of each node. Fig.3a, Fig.3b, Fig.3c, Fig.3d, and Fig.3e depict the F1 of each of the 17 nodes of each group from D1. The blue bars of histograms show the F1 of the nodes with the baseline method, while the orange bars show improvement in the F1 of each node after the application of FL. It is seen from Fig.3, that the application of FL improves F1 of anomaly detection models of all nodes from D1. The F1 of each of the nodes from D2 is depicted by Fig.4a, Fig.4b, Fig.4c, Fig.4d, and Fig.4e. It can be concluded from Fig.4, that the application of FL improves the anomaly detection performance of each node in all 5 groups from D2, hence authenticating **RQ1**.

Fig.5a, Fig.5b, Fig.5c, Fig.5d, and Fig.5e depict the AUC of each of the 17 nodes of each group from D1, whereas Fig.6a, Fig.6b, Fig.6c, Fig.6d, and Fig.6e show the AUC of each of the 17 nodes of each group from D2. The blue bars show the baseline AUC of each node, whereas the orange bars show the FL AUC of each node. Fig.5, and Fig.6, show that the application of FL improves the anomaly detection AUC of each node of D1 and D2, hence validating **RQ1**.

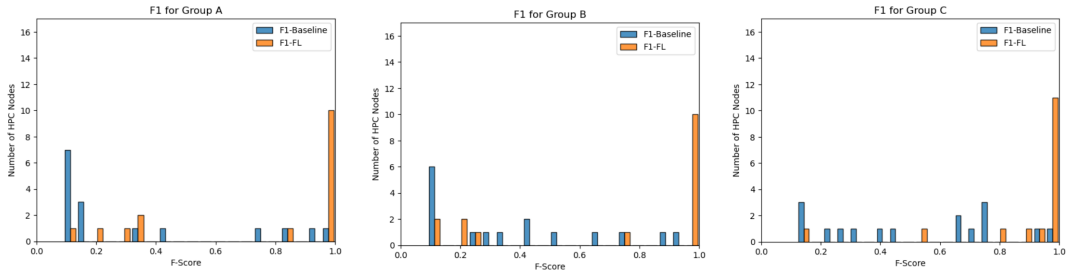
2) Reduced Training Set Experiment Results: **RQ2** is analyzed in this experiment. In the reduced training data experiment, the same FL methodology described in Sec.III-B is used – the only difference is the usage of a training set reduced by

TABLE IV: F1 of Reduced Training Set Sizes Obtained Over 5 Groups of 17 Nodes Each of D1 & D2

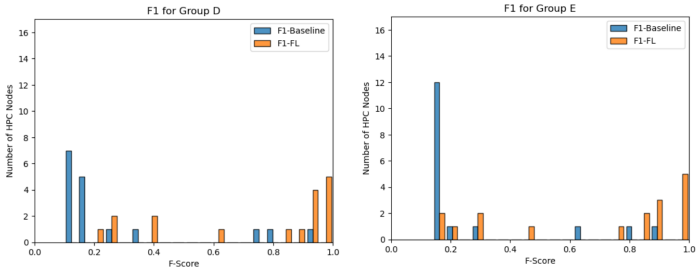
	F1 Full Data-Baseline	**F1 Full Data-FL**	**F1 $1/2^{th}$ Data-Baseline**	**F1 $1/2^{th}$ Data-FL**	**F1 $1/4^{th}$ Data-Baseline**	**F1 $1/4^{th}$ Data-FL**	**F1 $1/8^{th}$ Data-Baseline**	**F1 $1/8^{th}$ Data-FL**	**F1 $1/16^{th}$ Data-Baseline**	**F1 $1/16^{th}$ Data-FL**
	Data Set 1									
Group A	0.31	**0.86**	0.24	**0.65**	0.2	**0.45**	0.18	**0.4**	0.12	**0.35**
Group B	0.25	**0.78**	0.19	**0.59**	0.15	**0.51**	0.13	**0.39**	0.09	**0.33**
Group C	0.37	**0.87**	0.28	**0.64**	0.22	**0.57**	0.19	**0.49**	0.13	**0.42**
Group D	0.24	**0.84**	0.19	**0.65**	0.16	**0.53**	0.14	**0.34**	0.11	**0.29**
Group E	0.22	**0.79**	0.17	**0.57**	0.13	**0.48**	0.08	**0.33**	0.05	**0.28**
	Data Set 2									
Group A	0.54	**0.96**	0.45	**0.85**	0.36	**0.74**	0.25	**0.63**	0.18	**0.59**
Group B	0.42	**0.85**	0.38	**0.76**	0.29	**0.68**	0.18	**0.57**	0.09	**0.49**
Group C	0.22	**0.70**	0.14	**0.61**	0.09	**0.52**	0.06	**0.44**	0.03	**0.38**
Group D	0.33	**0.77**	0.25	**0.68**	0.16	**0.57**	0.09	**0.48**	0.05	**0.37**
Group E	0.31	**0.81**	0.24	**0.73**	0.15	**0.64**	0.08	**0.53**	0.04	**0.46**

TABLE V: AUC of Reduced Training Set Sizes Obtained Over 5 Groups of 17 Nodes Each of D1 & D2

	AUC Full Data-Baseline	**AUC Full Data-FL**	**AUC $1/2^{th}$ Data-Baseline**	**AUC $1/2^{th}$ Data-FL**	**AUC $1/4^{th}$ Data-Baseline**	**AUC $1/4^{th}$ Data-FL**	**AUC $1/8^{th}$ Data-Baseline**	**AUC $1/8^{th}$ Data-FL**	**AUC $1/16^{th}$ Data-Baseline**	**AUC $1/16^{th}$ Data-FL**
	Data Set 1									
Group A	0.38	**0.74**	0.25	**0.55**	0.17	**0.43**	0.14	**0.42**	0.08	**0.4**
Group B	0.44	**0.77**	0.31	**0.58**	0.21	**0.49**	0.16	**0.48**	0.12	**0.46**
Group C	0.47	**0.85**	0.35	**0.62**	0.26	**0.53**	0.22	**0.5**	0.18	**0.49**
Group D	0.33	**0.69**	0.27	**0.44**	0.14	**0.39**	0.11	**0.36**	0.07	**0.34**
Group E	0.25	**0.63**	0.18	**0.4**	0.09	**0.31**	0.05	**0.29**	0.01	**0.26**
	Data Set 2									
Group A	0.51	**0.94**	0.43	**0.85**	0.35	**0.73**	0.24	**0.59**	0.15	**0.52**
Group B	0.42	**0.82**	0.33	**0.73**	0.21	**0.6**	0.14	**0.51**	0.09	**0.43**
Group C	0.17	**0.67**	0.12	**0.59**	0.09	**0.48**	0.05	**0.39**	0.01	**0.29**
Group D	0.3	**0.8**	0.28	**0.69**	0.20	**0.57**	0.11	**0.42**	0.05	**0.35**
Group E	0.28	**0.78**	0.17	**0.67**	0.11	**0.56**	0.08	**0.45**	0.04	**0.36**



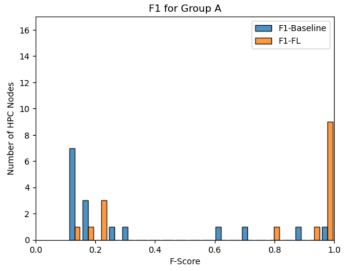
(a) Increase in F1 of Group A after FL (b) Increase in F1 of Group B after FL (c) Increase in F1 of Group C after FL



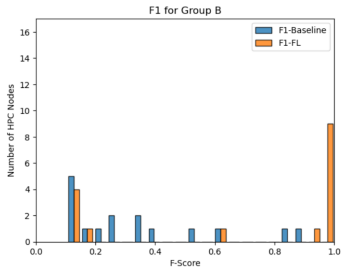
(d) Increase in F1 of Group D after FL (e) Increase in F1 of Group E after FL

Fig. 3: Individual Node Analysis for increase in F1 after FL of D1

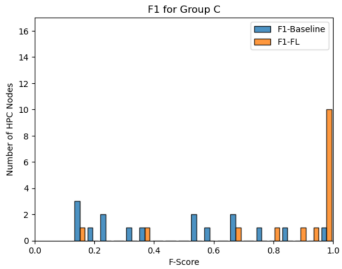
a factor r is used. Reducing the training set size would have a great impact on the applicability of anomaly detection method to real supercomputers, as the detection could be deployed more rapidly (as fewer training samples would be needed). We executed our experiments with reduced sizes of training data i.e. 1/2 (2.5 Months), $1/4^{th}$(1.25 Months), $1/6^{th}$(3.33 Weeks), $1/8^{th}$(2.5 Weeks) and $1/16^{th}$(1.25 Weeks). We wanted to find out what minimal quantity of training data would produce the same anomaly detection results as the baseline method with full training data.



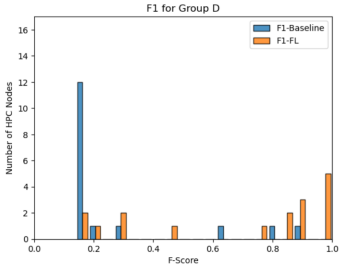
(a) Increase in F1 of Group A after FL



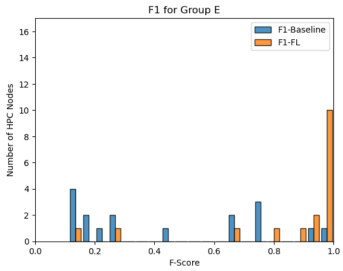
(b) Increase in F1 of Group B after FL



(c) Increase in F1 of Group C after FL

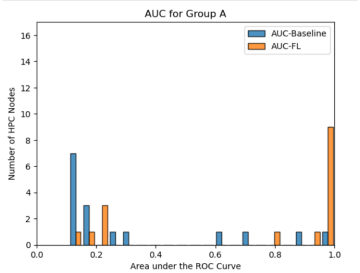


(d) Increase in F1 of Group D after FL

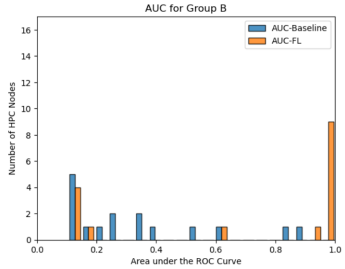


(e) Increase in F1 of Group E after FL

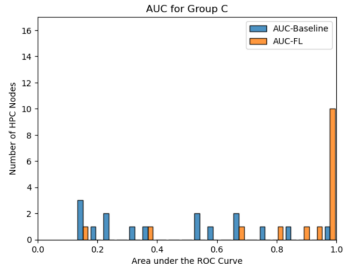
Fig. 4: Individual Node Analysis for increase in F1 after FL of D2



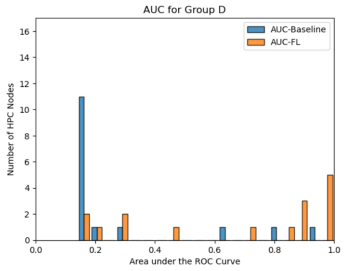
(a) Increase in AUC of Group A after FL



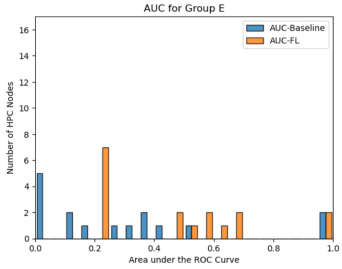
(b) Increase in AUC of Group B after FL



(c) Increase in AUC of Group C after FL



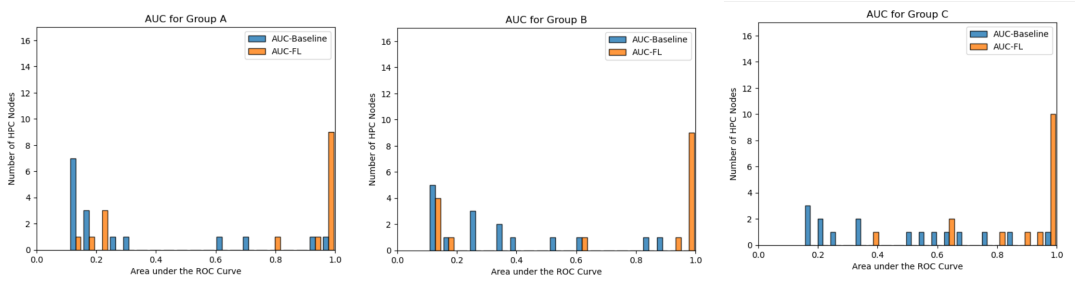
(d) Increase in AUC of Group D after FL



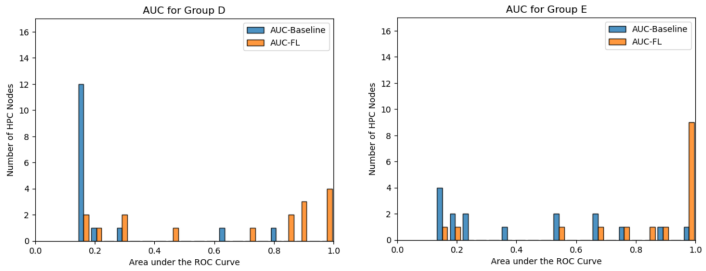
(e) Increase in AUC of Group E after FL

Fig. 5: Individual Node Analysis for increase in AUC after FL of D1

Table IV and Table V show the results of this experiment. The 4^{th}, 6^{th}, 8^{th} and 10^{th} columns of Table IV, depict the F1 of 5 groups of 17 nodes each, from D1 and D2, after applying FL on anomaly detection with reduced training data sizes. The application of F1 to anomaly detection with reduced training data sizes gives improved F1 than baseline non-FL anomaly detection models for full training data (shown in 1^{st} column of Table IV) for D1 and D2. Thus authenticating **RQ2**. In Table V, the 4^{th}, 6^{th}, 8^{th} and 10^{th} columns show the AUC of 5 groups of 17 nodes each, from D1 and D2, after applying FL on anomaly detection with reduced training data sizes. It can be concluded that the application of FL to reduced training data improves AUC compared to the baseline with full training set (shown in the 1^{st} column of Table V), hence validating **RQ2**.



(a) Increase in AUC of Group A after FL (b) Increase in AUC of Group B after FL (c) Increase in AUC of Group C after FL



(d) Increase in AUC of Group D after FL (e) Increase in AUC of Group E after FL

Fig. 6: Individual Node Analysis for increase in AUC after FL of D2

V. Conclusion

This research focuses on applying Federated Averaging to improve the effectiveness of unsupervised LSTM models for anomaly detection in HPC (High-Performance Computing) nodes. In this approach, each node is treated as a client within the Federated Learning (FL) framework. The method leverages data-driven models, supported by monitoring infrastructures that gather data to characterize the behavior of HPC systems—an integral feature of modern supercomputers. FL enables the enhancement of local models by utilizing shared behavioral similarities between HPC nodes, without the need for direct data sharing. The approach was validated using real historical data from a production-grade supercomputer, showing significant improvements in anomaly detection performance, notably in metrics like the F1-score and AUC. This marks the first time FL has been applied to this specific problem, with highly promising results.

Additionally, a significant reduction in the data collection period for reliable anomaly detection was noted. Traditional semi-supervised anomaly detection methods for HPC systems typically require around four months of normal operation data (i.e., data from periods without faults) for training. While this duration is manageable given the lifespan of HPC systems, which are closely monitored early on, reducing the training time would expedite model deployment and increase their effectiveness. With FL, we demonstrated that the data collection period can be reduced to just a few weeks, with performance comparable to four months of training without FL. This is due to the positive effect of aggregating information across multiple HPC nodes, improving each individual model after FL updates.

Future plans include deploying this FL strategy in real-world HPC environments and expanding the research by testing it on other supercomputers and nodes. We also aim to experiment with other FL approaches, such as FedAdam and FedAdagrad, to further enhance anomaly detection in HPC systems. Finally, we will explore alternative implementations for the FL approach, possibly using specialized FL libraries and tools tailored to specific hardware, such as MindSpore [10] for Huawei systems.

References

[1] H. Neau, R. Ansart, C. Baudry, Y. Fournier, N. Mérigoux, C. Koren, J. Laviéville, N. Renon, and O. Simonin, "Hpc challenges and opportunities of industrial-scale reactive fluidized bed simulation using meshes of several billion cells on the route of exascale," *Powder Technology*, p. 120018, 2024.

[2] L. Tunini, A. Magrin, G. Rossi, and D. Zuliani, "Global navigation satellite system (gnss) time series and velocities about a slowly convergent margin processed on high-performance computing (hpc) clusters: products and robustness evaluation," *Earth System Science Data*, vol. 16, no. 2, pp. 1083–1106, 2024.

[3] J. Wei, M. Chen, L. Wang, P. Ren, Y. Lei, Y. Qu, Q. Jiang, X. Dong, W. Wu, Q. Wang *et al.*, "Status, challenges and trends of data-intensive supercomputing," *CCF Transactions on High Performance Computing*, vol. 4, no. 2, pp. 211–230, 2022.

[4] B. Casella, R. Esposito, C. Cavazzoni, and M. Aldinucci, "Benchmarking fedavg and fedcurv for image classification tasks," *arXiv preprint arXiv:2303.17942*, 2023.

[5] X. Zhu, J. Cherukuri, and T. Yuan, "Failure and maintenance analysis of supercomputers," in *2016 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2016, pp. 1–6.

[10] https://www.mindspore.cn/en

[6] A. Borghesi, A. Burrello, and A. Bartolini, "Examon-x: a predictive maintenance framework for automatic monitoring in industrial iot systems," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 2995–3005, 2021.

[7] S. H. Rafique, A. Abdallah, N. S. Musa, and T. Murugan, "Machine learning and deep learning techniques for internet of things network anomaly detection—current research trends," *Sensors*, vol. 24, no. 6, p. 1968, 2024.

[8] S. Thudumu, P. Branch, J. Jin, and J. Singh, "A comprehensive survey of anomaly detection techniques for high dimensional big data," *Journal of Big Data*, vol. 7, pp. 1–30, 2020.

[9] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 634–644, 2019.

[10] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, and A. Borghesi, "A machine learning approach to online fault classification in hpc systems," *Future Generation Computer Systems*, vol. 110, pp. 1009–1022, 2020.

[11] M. Molan, A. Borghesi, D. Cesarini, L. Benini, and A. Bartolini, "Ruad: Unsupervised anomaly detection in hpc systems," *Future Generation Computer Systems*, vol. 141, pp. 542–554, 2023.

[12] B. Aksar, Y. Zhang, E. Ates, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, and A. K. Coskun, "Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems," in *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings 36*. Springer, 2021, pp. 195–214.

[13] A. Borghesi, M. Molan, M. Milano, and A. Bartolini, "Anomaly detection and anticipation in high performance computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 739–750, 2021.

[14] S. M. S. Bukhari, M. H. Zafar, M. Abou Houran, S. K. R. Moosavi, M. Mansoor, M. Muaaz, and F. Sanfilippo, "Secure and privacy-preserving intrusion detection in wireless sensor networks: Federated learning with scnn-bi-lstm for enhanced reliability," *Ad Hoc Networks*, vol. 155, p. 103407, 2024.

[15] A. Mora, A. Bujari, and P. Bellavista, "Enhancing generalization in federated learning with heterogeneous data: A comparative literature review," *Future Generation Computer Systems*, 2024.

[16] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2021.

[17] E. Farooq and A. Borghesi, "A federated learning approach for anomaly detection in high performance computing," in *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2023, pp. 496–500.

[18] H. Yadav and A. Thakkar, "Noa-lstm: An efficient lstm cell architecture for time series forecasting," *Expert Systems with Applications*, vol. 238, p. 122333, 2024.

[19] A. Cherif, A. Badhib, H. Ammar, S. Alshehri, M. Kalkatawi, and A. Imine, "Credit card fraud detection in the era of disruptive technologies: A systematic review," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 1, pp. 145–174, 2023.

[20] X. Wang, H. Wang, B. Bhandari, and L. Cheng, "Ai-empowered methods for smart energy consumption: A review of load forecasting, anomaly detection and demand response," *International Journal of Precision Engineering and Manufacturing-Green Technology*, vol. 11, no. 3, pp. 963–993, 2024.

[21] N. Jeffrey, Q. Tan, and J. R. Villar, "A hybrid methodology for anomaly detection in cyber–physical systems," *Neurocomputing*, vol. 568, p. 127068, 2024.

[22] J. Pei, K. Zhong, M. A. Jan, and J. Li, "Personalized federated learning framework for network traffic anomaly detection," *Computer Networks*, vol. 209, p. 108906, 2022.

[23] R. A. Sater and A. B. Hamza, "A federated learning approach to anomaly detection in smart buildings," *ACM Transactions on Internet of Things*, vol. 2, no. 4, pp. 1–23, 2021.

[24] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6348–6358, 2020.

[25] N. Sukhija, E. Bautista, D. Butz, and C. Whitney, "Towards anomaly detection for monitoring power consumption in hpc facilities," in *Proceedings of the 14th International Conference on Management of Digital EcoSystems*, 2022, pp. 1–8.

[26] B. Aksar, E. Sencan, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, B. Kulis, M. Egele, and A. K. Coskun, "Prodigy: Towards unsupervised anomaly detection in production hpc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–14.

[27] M. Molan, A. Borghesi, L. Benini, and A. Bartolini, "Analysing supercomputer nodes behaviour with the latent representation of deep learning models," in *European Conference on Parallel Processing*. Springer, 2022, pp. 171–185.

[28] P. Zou, A. Li, K. Barker, and R. Ge, "Detecting anomalous computation with rnns on gpu-accelerated hpc machines," in *Proceedings of the 49th International Conference on Parallel Processing*, 2020, pp. 1–11.

[29] K. Ott and R. Mahapatra, "Hardware performance counters for embedded software anomaly detection," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 528–535.

[30] M. Molan, M. S. Ardebili, J. A. Khan, F. Beneventi, D. Cesarini, A. Borghesi, and A. Bartolini, "Graafe: Graph anomaly anticipation framework for exascale hpc systems," *Future Generation Computer Systems*, 2024.

[31] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 9428–9433.

[32] A. Mora, D. Fantini, and P. Bellavista, "Federated learning algorithms with heterogeneous data distributions: An empirical evaluation," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE, 2022, pp. 336–341.

[33] J. Chang, K. Lu, Y. Guo, Y. Wang, Z. Zhao, L. Huang, H. Zhou, Y. Wang, F. Lei, and B. Zhang, "A survey of compute nodes with 100 tflops and beyond for supercomputers," *CCF Transactions on High Performance Computing*, pp. 1–20, 2024.

[34] A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, and C. Cavazzoni, "Paving the way toward energy-aware and automated datacentre," in *Workshop Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–8.

[35] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, vol. 2, 2016.

[36] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1038–1043.

[37] A. Bartolini, A. Borghesi, A. Libri, F. Beneventi, D. Gregori, S. Tinti, C. Gianfreda, and P. Altoè, "The davide big-data-powered fine-grain power and performance monitoring support," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, 2018, pp. 303–308.

[38] A. Borghesi, C. Di Santi, M. Molan, M. S. Ardebili, A. Mauri, M. Guarrasi, D. Galetti, M. Cestari, F. Barchi, L. Benini *et al.*, "M100 exadata: a data collection campaign on the cineca's marconi100 tier-0 supercomputer," *Scientific Data*, vol. 10, no. 1, p. 288, 2023.

[39] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon, "Towards a rigorous evaluation of time-series anomaly detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 7, 2022, pp. 7194–7201.