A Federated Learning Approach for Anomaly Detection in High Performance Computing

A Federated Learning Approach for Anomaly Detection in High-Performance Computing

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 complex machines that need to be operated at their maximum potential to recoup their investment cost and to mitigate their environmental impact. Anomalous conditions hindering the correct usage of the supercomputing nodes are a significant problem. Hence, the development of automated anomaly detection techniques remains a vital area of research. Machine Learning (ML) models demonstrated to be good at detecting anomalies on individual nodes. However, the potential of combining data from multiple computing nodes and associated ML models has not been explored yet. Federated Learning (FL) can address this shortcoming, by allowing individual models to learn from each other. This paper applies FL to improve the performance of anomaly detection models for HPC systems. The approach has been validated on data from an actual supercomputer, obtaining an improvement in the average f-score from 0.31 to 0.84. We also show how FL can significantly shorten the data collection period needed to create a training set. While ML models need, on average, 4.5 months of training data, FL reduces the training set size to 1.2 weeks – a 15x reduction.**

***Index Terms*—Federated Learning, High Performance Computing, Anomaly Detection, Machine Learning**

I. Introduction

High Performance Computing (HPC) systems, data centers, and warehouse-scale computing are vital aspects of today's industry and society [1]. They comprise several computing rooms, each having many racks with tens/hundreds of computing nodes. Each node is powered by heterogeneous processing units including CPUs and accelerators (e.g. GPU, NPU, FPGA); users submit jobs running as parallel applications. Supercomputers are becoming increasingly sophisticated, as the requirements for computer performance also increase. A very significant issue is the detection of fault conditions. Historically, system management was done using scripts and the intervention of system administrators; automated tools to detect anomalies in HPC nodes would be extremely beneficial [2]. Log-based analysis has been typically conducted post-mortem to understand the cause of failures that have happened [3]. Recently, another research line exploited the data collected from current supercomputers [4], which HW sensors and SW probes to monitor the status of the different components.

The authors would like to thank Huawei for the availability of their hardware and support. This work has been partially supported by European Project HORIZON-CL4-2021-HUMAN-01-AI4Europe (g.a. 101070000) and EU Horizon 2020 Project StairwAI (g.a. 101017142). We also thank ARPAE, Emilia Romagna, Bologna.

Most data-driven approaches drew inspiration from Machine Learning (ML) and Deep Learning (DL) domain [5], including supervised approaches that assume labeled training data [6] and unsupervised ones [7]. Semi-supervised models have been demonstrated to be very good at merging the strengths from both areas' weaknesses [8]. These models are tightly connected to a specific HPC node, that is the HPC node whose data has been used for training. However, the node failures tend to be comparable and to follow similar patterns. Hence, combining information from multiple HPC nodes can aid automated anomaly detection; for this scope, we employ Federated Learning (FL) [9]. While FL has been used for anomaly detection [9], this is the first application to the HPC domain. We focus on semi-supervised models for anomaly detection of HPC systems, as they have been proven to be a good practical trade-off between detection accuracy and less stringent requirements in terms of available annotated data. We demonstrate the effectiveness of our approach by validating it on data coming from a real HPC system[1].

The main contributions of this paper are the following:

- We apply FL to enhance semi-supervised anomaly detection models for HPC nodes (this is the first application of FL in this domain); we observe a steep improvement in detection accuracy, with average f-score rising from 0.31 (no FL) to 0.84 (using FL).
- We additionally demonstrate how FL can reduce the training set size needed to train the anomaly detection models – while the non-FL approach requires 4.5 months of training data to obtain high accuracy, the FL approach has equivalent results in just 1.2 weeks.

II. Related Works

In HPC, anomalies are time periods in which computing jobs turn out to be incomplete or erroneous, affecting the quality of computing services provided to the users. The vast majority of data generated by supercomputers are non-anomalous. Thus due to unbalance in data, typical supervised anomaly detection approaches are not well-suited without the adoption of data re-balancing mechanisms. Additionally, annotated data is not always available; this is only partially

[1]Namely Marconi100, hosted at CINECA facilities – the Italian supercomputing center – in Bologna. https://www.hpc.cineca.it/hardware/marconi100

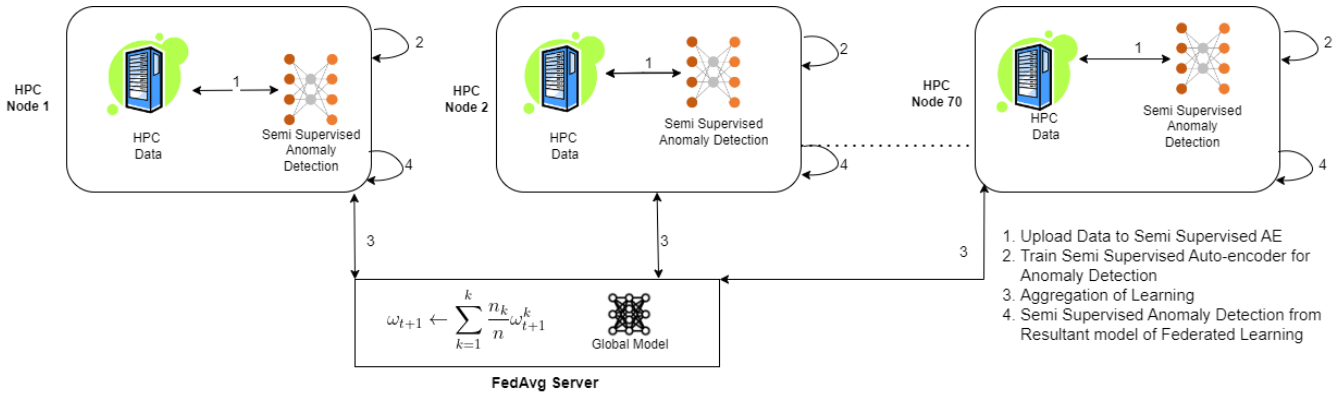

Fig. 1. Semi-Supervised Anomaly Detection for HPC using FL

mitigated by SW tools recording the status of HPC nodes, such as Nagios[2]. Thus, supervised approaches remain not applicable to all HPC contexts. Recently, semi-supervised anomaly detection techniques have been proposed; they exploit the fact that the most of data is normal [10]. A DL model called autoencoder is used to learn the healthy state of a supercomputer. The autoencoder is trained using only normal samples; it learns the normal behavior in its latent state. At test time the reconstruction error is used to discriminate between normal and anomalous data. In all cases, a different DL model is needed for each computing node. FL has been used to facilitate anomaly detection in several disciplines, from network anomaly detection [9] to industrial settings [11]. FL allows DL models to train and learn from each other without sharing their data. To the best of our knowledge, this is the first FL application to anomaly detection in HPC.

III. Methodology

The goal of our approach is to improve the automated anomaly detection of HPC nodes using FL. As this is the first attempt at employing FL for this task, we decided to focus on a specific approach for anomaly detection in HPC, namely the semi-supervised method, as it has demonstrated very good results when applied to real HPC while not requiring fully annotated data (which is not always available in the HPC context); in future works, we plan to explore the impact of FL to other underlying ML models for anomaly detection. We start with a key assumption (the same described in previous semi-supervised approaches for HPC anomaly detection, see [5], [10]), that is we suppose that the training data will contain only examples corresponding to normal operating condition; this is a safe assumption as it is typically possible to identify relatively long healthy periods right after the installation of a new supercomputer before the HW components start to degrade. To allow for a fair comparison (FL usage against no FL) we opted to use as baseline ML model the same semi-supervised deep neural network (DNN) employed in [10][3], namely an autoencoder network.

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

[3]The current state-of-the-art for semi-supervised anomaly detection in HPC

Previous works explained how to train an autoencoder for each computing node, and how to use a threshold mechanism to detect anomalies at inference time – the autoencoders for the different nodes were trained separately. Instead, in this work the autoencoders are trained in a federated manner, improving their prediction performance by learning from other models. Fig. 1 shows the workflow of the approach.

A. Semi-Supervised Anomaly Detection

In this step, an autoencoder DNN is trained only on non-anomalous data. The autoencoder is made to learn the healthy behavior of the system. Then, the network can be used for anomaly detection by analyzing its reconstruction error for new data [12]. If the autoencoder reconstruction error of a new value is greater than a threshold, the new value is declared as an anomaly, otherwise a normal value. The reconstruction error of an autoencoder is defined as the mean error over all features in the data. The threshold is chosen by observing the errors distribution in the training set and by selecting the n-th percentile; the value of n is a crucial hyperparameter. In this work we relied on the guidelines provided by the state-of-the-art to identify good initial candidates [10], and then we performed a manual fine-tuning. While impactful, the selection of the optimal threshold is a problem orthogonal to the one at hand; in this work, we focus on the benefits due to FL, and for a fair comparison we keep the same threshold for both the FL and non-FL experiments.

B. The Federated Learning Approach

The motivation for incorporating FL is that the autoencoder models previously described are trained independently. However, the single models might benefit from joint learning [13], as the behavior of the nodes is similar, albeit not identical. The architecture of FL architecture is reported in Fig. 1. There are two vital components: a central server (this role can be performed by the login/management node that in typical HPC systems hosts the resource manager and the job scheduler) and autoencoder models as clients; the clients are the computing nodes, which communicate with the management nodes (i.e., receive the workload to be executed); each computing node has

a daemon continuously running and collecting node-related information, with minimal overhead [4], that produces the data used for training and then to perform anomaly detection on live streams. Each client has an autoencoder model that has been trained on an HPC node. To start FL, the server sends weights to the clients. After one or more training steps, each client sends its weights to the central server. The server updates the global model by aggregating all weights received from the clients [11]. The central server uses the FedAvg algorithm [11], to do a weighted average of weights obtained from all clients as shown in Eq. 1.

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

Here ω_{t+1} is the most recent weights sent to clients by the server. *t+1* is the recent round of communication. ω_{t+1}^k is the weights sent by client *k* to server in the *t+1* round of communication. *n* is the total number of data values used for training the global model, while n_k is the number of data samples used by client *k* for local training [14]. At the end of FL, all autoencoder clients have been updated, with global model. Each of them performs semi-supervised anomaly detection again, on their respective HPC node. In the reduced training data size experiment, the methodology remains the same; the only difference is that a reduced training set size is used to train the client models (the training set is reduced by a factor r).

IV. Experimental Analysis

This research was executed in a Ubuntu 22.0 environment. The system has 790 GB of RAM. The system also has 42 Intel(R) Xeon(R) Gold 6240R CPUs with a total of 96 cores.

A. Data

The data used in this research is from a tier-0 production supercomputer, Marconi100, hosted at CINECA, Bologna. The current system has more 980 nodes. Marconi100 has been the flagship HPC system until 2023, peaking at number #9 in the Top500 list in 2020. Each node has 2x16 cores (IBM POWER9 AC922 at 3.1 GHz) and 4 Nvidia Volta V100 GPUs, plus 256 GB of RAM. The data has sensor measures from hardware (HW) sensors and program counters, ranging from the power consumption of HW devices such as GPUs, CPUs, RAM memory, etc., to clock frequency and fan speed. In this work, we focus on node-related metrics as the aim is to detect anomalies at the node level. The data used in this research also has information about a node's status, extracted using the software tool Nagios. This tool is used by system administrators to flag nodes that are behaving wrongly and that should be the subject of maintenance operations. The Nagios-generated information is used to annotate the data. Nagios has a 15-minute node-labeling rate, hence we employ the same 15-minute aggregation window.

Average value and standard deviation are computed over the time window, thus incorporating the temporal dynamics of the collected metrics. The label (the target for our anomaly detection task) can be either zero (normal operation) or non-zero, fault condition. As this is the first evaluation of the FL approach in HPC, we decided to validate it on a sub-set of nodes, hence we randomly chose 70 nodes to conduct our experiments; we note that this number is sufficiently high to ensure that our results are not due to random chance. Table I depicts the average of normal, anomalous data points and the percentage of anomalies across all 70 nodes. Anomalies are scarce and happen randomly in the data (they do not follow any specific distribution); this scarcity is one of the reasons why supervised approaches are typically not very effective on real data, as the data is very unbalanced.

TABLE I
Average Number of Normal, Anomalous, Total Data Points and Average Number of Anomalies of all 70 nodes

Normal	Anomalous	Total	% Anom
12650.25	391.25	13041.49	3.00

B. Approach Implementation

There is one autoencoder model for each of the 70 HPC nodes, each trained in a semi-supervised fashion. There are 461 numerical features in the data, that were normalized in the 0 to 1 range, using scikit-learn[4]. The data for each node is randomly split, 80% used for training and 20% for testing. After this split, anomalous data were removed from the training set and moved to the test set. We used 10 epochs for training all autoencoder models and a batch size equal to 10. The input layer to the auto-encoder had a size of 460. There were three successive dense layers of size 80, 60 and 40 respectively. The hidden layer had a size of 20 neurons. It is a dense layer with a ReLU activation function. The decoder has 4 dense layers of sizes 40, 60, 80, and 100 respectively. All hyperparameters were chosen after a preliminary empirical evaluation, guided by the study of the literature. The loss function was root mean squared error. The autoencoder has been trained using RMProp optimizer, with a learning rate of 0.000001. The reconstruction error is used to train the autoencoder (minimizing it), and then to detect anomalies as well. Since the autoencoders learned to recognize normal points (being the only examples shown during training), a low reconstruction error at test time indicates a normal sample and a higher error reveals an anomaly, instead [10]. To distinguish anomalies from normal points, the reconstruction error must be compared to a threshold, identified as the n-th percentile computed over the training set; we selected the 98^{th} percentile. Each autoencoder was implemented with Tensorflow.

Flower[5]was used to implement the FL strategy. Namely, we have clients (a Python script) that train the autoencoder models (one per node) participating in the FedAvg strategy. In addition, there is also a FedAvg server entity that coordinates the interaction. We experimented with 1000, 1500 and 2000

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

[5]https://flower.dev/

TABLE II
EXPERIMENTAL RESULTS OBTAINED OVER 70 NODES

	Prec.-AE	Prec.-FL	Recall-AE	Recall-FL	F1-AE	F1-FL	TNR-AE	TNR-FL	FPR-AE	FPR-FL
Avg.	0.24+/-0.31	0.79+/-0.30	1	1	0.31+/-0.33	0.84+/-0.27	0.24+/-0.30	0.81+/-0.32	0.76+/-0.33	0.19+/-0.29

rounds of FL (following the literature [14]), but no significant performance changes were noted. At the end of FL, each node has the resultant model of FL. All nodes perform semi-supervised anomaly detection again with the updated model.

C. Results

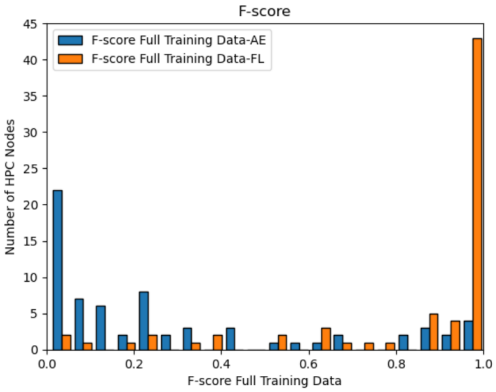

Fig. 2. Histogram for Improvement in F-score of Full Training Data after FL

The metrics calculated in this research are precision ($Precision = \frac{TP}{TP+FP}$), recall ($Recall = \frac{TP}{TP+FN}$), f-score ($FScore = 2 * \frac{Precision*Recall}{Precision+Recall}$), true positive rate ($TPR = \frac{TP}{TP+FN}$), true negative rate ($TNR = \frac{TN}{TN+FP}$), false positive rate($FPR = \frac{FP}{TN+FP}$), and false negative rate ($FNR = \frac{FN}{TP+FN}$) [12].

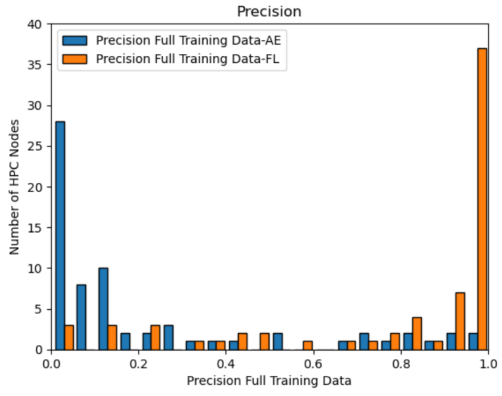

Fig. 3. Histogram for Improvement in Precision of Full Training Data after FL

As depicted in Table II, the average precision of 70 nodes with baseline autoencoder is 0.24 and the average f-score is 0.31. The recall of the overall experimental setup is 1. It is a threshold-based method hence the autoencoder does not allow any anomaly to be missed. The validity of recall being 1 in this data set has been manually verified from predictions of 25 nodes to validate that the autoencoder is correctly returning a recall of 1. The average precision of all HPC nodes improves

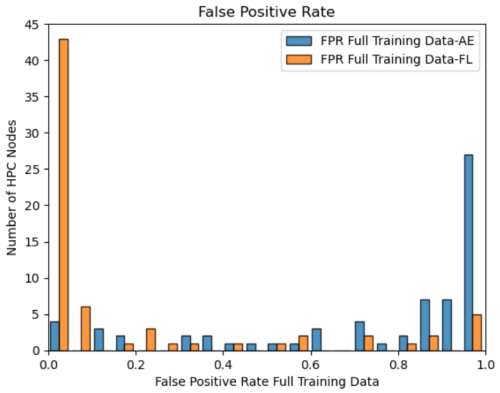

Fig. 4. Histogram for Decline in False Positive Rate of Full Training Data after FL

to 0.79 and the average f-score improves to 0.84 after FL, as shown in Table II. Table II depicts that the average true negative rate improves to 0.81 and the average false positive rate drops to 0.19 after FL. The true positive rate is always 1. The false negative rate is always 0. The f-score of more than 40 nodes increases to the 0.95 to 1 interval as shown in Figure 2. The precision of more than 35 HPC nodes reaches the 0.95 to 1 range, after FL as shown in Figure 3. Further Fig. 4 depicts that the false positive rate of more than 40 nodes drops to the 0 to 0.05 range after FL.

In our dataset, there are an average of 12,865 data values per node. Each data value is calculated after 15 minutes, which is equivalent to 4.5 months of data to effectively train deep learning models for anomaly detection. In another experiment, we wanted to analyze if the learning capabilities of FL can be utilized to improve the prediction abilities of models with less training data [15]. To evaluate this in the context of this research, the above experiment was executed out with $1/16^{th}$, $1/8^{th}$, $1/6^{th}$, $1/4^{th}$, and 1/2 of training data. The goal of the evaluation was to see what least amount of training data, with FL, would generate the same prediction results as the baseline autoencoder on full training data. From Table III, it can be seen that the average precision of the autoencoder of full training data is 0.24. The average precision with FL training on $1/16^{th}$ training data is 0.51 and 0.60 on $1/8^{th}$ training data. FL training gives 0.64 precision on $1/6^{th}$, and $1/4^{th}$ training data. Lastly, Table III also shows that FL training gives 0.69 precision on 1/2 training data.

Table III depicts that with $1/16^{th}$ training data, the average

TABLE III
EXPERIMENTAL RESULTS OF REDUCED TRAINING SET SIZES OBTAINED OVER 70 NODES

	Prec.-AE	**Prec.-FL**	**Recall-AE**	**Recall-FL**	**F1-AE**	**F1-FL**	**TNR-AE**	**TNR-FL**	**FPR-AE**	**FPR-FL**
Avg. - Full Data	0.24+/-0.31	0.79+/-0.30	1	1	0.31+/-0.33	0.84+/-0.27	0.24+/-0.30	0.81+/-0.32	0.76+/-0.33	0.19+/-0.29
Average - 1/2 Data	0.23+/-0.30	0.69+/-0.32	1	1	0.29+/-0.32	0.75+/-0.31	0.25+/-0.31	0.64+/-0.38	0.75+/-0.29	0.35+/-0.32
Avg. - $1/4^{th}$ Data	0.21+/-0.28	0.64+/-0.34	1	1	0.28+/-0.31	0.71+/-0.32	0.22+/-0.29	0.63+/-0.39	0.78+/-0.31	0.37+/-0.34
Avg. - $1/6^{th}$ Data	0.23+/-0.31	0.64+/-0.36	1	1	0.29+/-0.33	0.70+/-0.34	0.25+/-31	0.55+/-0.40	0.75+/-0.33	0.45+/-0.34
Avg. - $1/8^{th}$ Data	0.22+/-0.29	0.60+/-0.37	1	1	0.29+/-0.31	0.67+/-0.35	0.24+/-0.33	0.51+/-0.39	0.76+/-0.34	0.49+/-0.32
Avg. - $1/16^{th}$	0.25+/-0.310	0.51+/-0.39	1	1	0.22+/-0.34	0.58+/-0.38	0.23+/-0.30	0.47+/-0.39	0.77+/-0.36	0.53+/-0.35

f-score after FL is 0.58, and 0.67 with $1/8^{th}$ training data. FL with $1/6^{th}$ training data gives an average f-score of 0.70 and an average f-score of 0.71 with $1/4^{th}$ training data. The average f-score after FL is 0.75 with 1/2 training data. From Table III, it can be seen that the average true negative rate after FL is 0.47 with With $1/16^{th}$ training data, 0.51 with $1/8^{th}$ training data, 0.55 with $1/6^{th}$ training data, 0.63 with $1/4^{th}$ training data and 0.64 with 1/2 training data. It can also be seen that the average false negative rate after FL is 0.53 with $1/16^{th}$ training data, 0.49 with $1/8^{th}$ training data, 0.45 with $1/6^{th}$ training data, 0.37 with $1/4^{th}$ training data and 0.35 with 1/2 training data.

V. CONCLUSION

This research presents the application of FedAvg to improve the anomaly detection performance of semi-supervised models on HPC nodes. FL has been used to enrich local models by exploiting the underlying similarity of the behavior of the HPC nodes, without explicitly sharing data. The empirical analysis reveals how FL has the potential to be very beneficial to the anomaly detection task, as highlighted by a very marked improvement in overall relevant metrics. This is the first application of FL to this problem and the results are very promising. We also noticed a significant decrease in terms of data collection time required to obtain reasonably good results for anomaly detection. Without using FL, the state-of-the-art for semi-supervised anomaly detection in supercomputing systems requires an amount of training data equivalent to approximately 4 months of normal operation. However, a smaller training time would allow for quicker deployment of anomaly detection models, greatly improving their efficacy and benefits. We demonstrate that with FL it is possible to decrease the initial data collection time to just a few weeks (a couple of weeks are sufficient to obtain results on par with 4 months without FL). This happens as the aggregation of information from multiple HPC nodes has a positive effect on each individual model, after the FL-guide updates. In the future, we intend to deploy this FL strategy on real HPC systems. We also intend to try other FL approaches like FedAdam, and FedAdagrad on anomaly detection in HPC systems. Finally, we will consider other implementation options for the FL approach like using other FL libraries (including specialized tools for the hardware used for performing the experiments[6]).

REFERENCES

[1] ETP4HPC, "Strategic research agenda," 2017. [Online]. Available: http://www.etp4hpc.eu/sra.html/

[2] D. Mulero-Pérez, M. Benavent-Lledó, and et al., "Anomaly detection and virtual reality visualisation in supercomputers," *The International Journal of Advanced Manufacturing Technology*, pp. 1–13, 2023.

[3] E. Chuah, A. Jhumka, M. Malek, and N. Suri, "A survey of log-correlation tools for failure diagnosis and prediction in cluster systems," *IEEE Access*, vol. 10, pp. 133 487–133 503, 2022.

[4] A. Bartolini, F. Beneventi, and et al., "Paving the way toward energy-aware and automated datacentre," in *Proc. of the 48th International Conference on Parallel Processing: Workshops*, ser. ICPP 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–8.

[5] B. Aksar, Y. Zhang, and et al., "Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems," in *HPC: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proc. 36*. Springer, 2021, pp. 195–214.

[6] 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*, 2019.

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

[8] A. Borghesi, M. Molan, and et al., "Anomaly detection and anticipation in high performance computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 739–750, 2022.

[9] 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.

[10] A. Borghesi, A. Bartolini, and et al., "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of AI*, vol. 85, pp. 634–644, 2019.

[11] H. T. Truong, B. P. Ta, Q. A. Le, and et al., "Light-weight federated learning-based anomaly detection for time-series data in industrial control systems," *Computers in Industry*, vol. 140, p. 103692, 2022.

[12] A. Borghesi, M. Molan, and et al., "Anomaly detection and anticipation in high performance computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 739–750, 2022.

[13] A. Raza, K. P. Tran, L. Koehl, and S. Li, "Anofed: Adaptive anomaly detection for digital health using transformer-based federated learning and support vector data description," *Engineering Applications of Artificial Intelligence*, vol. 121, p. 106051, 2023.

[14] J. Toldinas, A. Venčkauskas, A. Liutkevičius, and N. Morkevičius, "Framing network flow for anomaly detection using image recognition and federated learning," *Electronics*, vol. 11, no. 19, p. 3138, 2022.

[15] L. Chou, Z. Liu, and et al., "Efficient and less centralized federated learning," in *ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*. Springer, 2021, pp. 772–787.

[6]For instance, using MindSpore (https://www.mindspore.cn/en) for Huawei-provided hardware.