Received 17 November 2023, accepted 28 November 2023, date of publication 5 December 2023,
date of current version 13 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3339556

RESEARCH ARTICLE

LSTM-Autoencoder-Based Incremental Learning for Industrial Internet of Things

ATALLO KASSAW TAKELE AND BALÁZS VILLÁNYI
Department of Electronics Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, 1117 Budapest, Hungary

Corresponding author: Atallo Kassaw Takele (atallo.takele@edu.bme.hu)

ABSTRACT Edge-based intelligent data analytics supports the Industrial Internet of Things (IIoT) to enable efficient manufacturing. Incremental learning in the edge-based data analytics has the potential to analyze continuously collected real-time data. However, additional efforts are needed to address performance, latency, resource utilization and storage of historical data challenges. This paper introduces an incremental learning approach based on Long-Short Term Memory (LSTM) autoencoders, by sparsening the weight matrix and taking samples from previously trained sub-datasets. The aim is to minimize the resources utilized while training redundant knowledge for edge devices of IIoT. The degree of sparsity can be determined by the redundancy of patterns, and the inverse of the coefficient of variation has been utilized to recognize it. A higher value of the inverse of the coefficient of variation shows that the values of the weight matrix are close to each other, which indicates the redundancy of knowledge, and vice versa. In addition, the coefficient of variation has been applied for limiting the size of samples from the previously trained sub-datasets. The experiment conducted using the IIoT testbed dataset demonstrates substantial enhancements in resource optimization without compromising performance.

INDEX TERMS Industrial internet of things (IIoT), incremental learning, LSTM-autoencoder, weight sparsification.

I. INTRODUCTION

The network of computing devices including sensors, intelligent communication gateway, controlling tiny devices and large servers bring a new term known as Internet of Things (IoT). The rapid growth of IoT touches several areas of application to enhance their performance, such as healthcare, home automation, transportation, industrial automation, agriculture and power plants [1]. The Industrial Internet of Things (IIoT) is a sub-category of IoT which is utilized in industrial scenarios. The IIoT is enabled by intelligence data analytics, sophisticated application software, controllers, sensors, actuators, and communication gateways [2], [3]. The IIoT devices deployed in the industrial environment collect information and transmit it to the cloud or edge server for subsequent analysis. The IIoT facilitates the intelligent manufacturing by improving safety, reducing downtime, visualization, and accurate decision making [3].

Now a days industries are using intelligence data analytics with smart devices for improving their safety, reducing downtime and fast decision making [4], [5]. The traditional way of data analytics was centralized approach which applies machine and deep learning algorithms to the massive data collected from several edge devices using a computationally reach server or cloud. However, the exchange of data from the manufacturing devices to the server causes security risk, communication cost and delay. Decentralized learning has been proposed to enable manufacturing devices to independently manage data analytics. The decentralized approach gets more attention by several scholars due to its ability to protect data security [6]. However, the adaptation of decentralized learning isn't straightforward and without limitations. It is limited by several bottlenecks like, resource limitation to store and analyze massive self-collected data. Although machine learning algorithms are effective in analyzing industrial devices

The associate editor coordinating the review of this manuscript and approving it for publication was Stefano Scanzio.

data, they are computationally cost processes. Hence, optimizing these sophisticated costly intelligent algorithms for resource limited devices is a challenge, which is the focus of this study.

Industrial activities are real-time processes in which devices generate time series data. Sequential algorithms are popular methods to effectively predict and forecast time series data. A Recurrent Neural Network (RNN) is among the most popular sequential models for analyzing temporal and sequential data [11], [27]. An RNN is a special type of neural network that is capable of memorizing the previous event. It is capable of doing sequence classification, time series forecasting, and sequence to sequence prediction [10], [12]. It has a number of variants in order to improve some of the shortcomings. Long Short-Term Memory (LSTM) is one of the most widely used variants of RNN that enhances the issue of long-term dependency. It is applied in several domains such as time series forecasting, speech recognition, natural language processing, predictive maintenance, predicting sensor data, and time series anomaly detection [10]. Traditional training was through feeding the available dataset to the LSTM network at once [14]. However, industrial devices produce data over time which needs continuous model updates. Incremental learning has been proposed for such a scenario by continuously transferring knowledge from the previously trained dataset.

Most of real-time applications (i.e., anomaly detection, automated driving, predictive maintenance) scenarios can't be solved by traditional models. As a result, it needs continuous updates in order to adapt to these complex scenarios. Human beings are able to recognize new features from unseen observation. Incremental learning is a dynamic approach that aims to update the performance over time using continuously collected data. Incremental learning is supposed to learn new insights from continuously collected data [19]. It is able to learn new classes from new observations, instead of traditional fixed classes. It also offers optimized utilization of computational time, memory, and energy requirements, as training on the entire dataset observed over a long period of time demands a significantly higher amount of resource [16]. Recent incremental learning approaches are also reducing catastrophic forgetting which is one of the main challenges of neural networks and it is caused by forgetting the past learned information with the arrival of new data [7]. Hence, the main objective of incremental learning is an efficient training of continuously observed real-time data with limited memory, computational time, and energy without losing performance [15], [19].

The incremental learning can be categorized into three groups based on the dataset they utilize [7], [26]. Namely, training without old data, using artificially generated data, and using samples of old data. In the first case old class data is not required while training the current task. As clearly examined the number of training data is small as it uses the current task data only which affects the performance of the prediction. Thus, it is trained by modifying the network structure and normalizing the parameters [26]. In the second case, artificial data which represents the old data could be generated using generative models. This improves the memory storage space required for storing old data, especially for memory-constrained devices. The artificial data is generated at runtime while training each task for improving performance. The performance of prediction depends on the ability of the generative model to produce better representative samples [7], [23]. The last option takes a sample dataset from each old training task which takes advantage of real old data. It maintains performance by formulating a representative dataset from the old and new tasks.

In deep LSTM networks a large number of parameters negatively affect the performance, require a higher computational resource, and take longer time to process [24]. Hence, a significant amount of storage, memory, bandwidth, and computational resources are required for deploying the LSTM model. It is a challenging task to regularize deep LSTMs, which can comprise millions of parameters and are susceptible to overfitting. In real-time applications, the strict real-time latency limitations make deep LSTMs impractical due to their latency. Common approaches for minimizing these dense deep networks are dropout and weight masking (matrix sparsification). Weight masking is to modify or exclude specific parameters (weights) within a neural network during training. This technique allows for selective adjustment of the weights to address problems such as minimizing resources, overfitting, and focusing on specific features. It includes multiplying the weights of a neural network by a binary mask that can be learned during training or predetermined. Matrix sparsification is removing or setting to zero a specific proportion of the network weights that have minimal impact on its output [18]. By using matrix sparsification, the total number of parameters in the network is decreased, leading to improved efficiency in terms of computational resources and storage.

Redundant knowledge in neural networks is a common issue and is characterized by the existence of duplicated (overlapping) information in the model, which can result in prolonged training periods, a decrease in generalization performance, and higher computational expenses [26]. It may be caused by the result of several factors such as incorporating redundant features or the existence of duplicated hidden units or weights in the network. Dealing with this problem enhances the effectiveness and efficiency of the neural network, as it can prevent decreasing model complexity, avoid overfitting, and optimize the utilization of computational resources. This is a serious issue, especially for small IoT devices deployed in different applications since they have limited computational resources and memory. Several approaches have been proposed to improve redundant knowledge in neural networks, such as model compression and weight sparsification [17], [24], [26].

This study focuses on weight sparsification and sampling of old data for tiny devices of IIoT using LSTM-autoencoder based incremental learning in order to enhance resource utilization. This proposed approach sparsens a certain level of the weight matrix for the on-device incremental learning methods. The amount of sparsity depends on the redundancy of knowledge and this paper applies the inverse of the coefficient of variation for examining the redundancy of information for the weight matrix. A higher value of the inverse of the coefficient of variation indicates that the values in the weight matrix are closely grouped together which suggests redundancy of knowledge. Conversely, a lower value suggests less redundancy. The second objective of the study is to propose a dynamic sampling mechanism for the old training dataset. This sampling technique also uses the coefficient of variation for determining the number of samples taken from each sub-dataset. Extensive experiments prove the effectiveness of the proposed approach by reducing computational resources.

The paper's contribution is summarized as follows:

- This study proposed an optimized LSTM-Autoencoder network using matrix sparsification and taking sample of previously processed data
- We utilized the data variation to determine the amount of weight to be sparsened and the percentage of the sample to be extracted from the previously processed data.
- We evenly distributed the weight to be sparsened across each row to minimize bias in the sparsening process

The remaining part of the paper is presented as follows: Section II details the background and related works in the proposed specific area, and Section III presents the methodology of the proposed approach. Section IV discusses about the experimental demonstration process and results, and Section V presents the conclusion of the proposed work.

II. BACKGROUND

A. EDGE COMPUTING AND INCREMENTAL LEARNING

Edge computing is a computing approach that involves performing computation and storing data in closer to the site (end node), instead of depending on a central location (edge server, cloud server) [20], [21]. It involves carrying out computing tasks at the network's edge, typically on a device situated in close proximity to the origin of the data. Devices such as sensors, actuators, PDAs, smart devices, and controlling devices are among the equipment that can be utilized in this computing paradigm. The need for real-time processing and analysis of data and the rise of IoT devices promotes the popularity of edge computing. Closer computation to the source of data preserves bandwidth, reduces latency, improves security, and faster processing.

Application of edge computing includes industrial automation, smart healthcare, autonomous vehicles, and smart cities. The utilization of edge computing in manufacturing and other industrial processes can lead to improved security, efficiency, and minimized downtime through the processing and analysis of data generated by machines [25]. In the autonomous vehicle, it enables the processing of huge amounts of data produced by sensors in real time, thereby facilitating quick and secure decision-making. Medical data produced by sensors and wearable devices could be analyzed locally and used in medical intervention and treatment. Smart cities nowadays are supported by edge processing for enhancing the quality of life and public safety using sophisticated equipments such as security cameras, water quality sensors, and traffic light sensors.

However, edge computing is hampered by several challenges such as computational cost, battery consumption, scalability, and interoperability [21]. The design of edge computing systems should be capable of managing the considerable amounts of data generated by IoT devices, while also being scalable to the evolving needs of the system. The integration of devices and systems from various vendors in edge computing systems can pose a challenge in ensuring that they function together cohesively. The battery-powered nature of most edge computing devices, combined with the substantial power consumption of processing large data volumes, can result in a lower battery lifespan. This challenge is more serious for devices deployed in remote and inaccessible areas. Data analysis supported by intelligent machine learning is very expensive which leads to higher computation costs. The optimization of real-time data analysis algorithms is an open issue for most applications. Incremental learning is a promising approach for resource constraint devices as the training is held using a branch of real-time instead of feeding huge data at once. On-device learning is an edge computing-related topic that refers to the ability of devices to train machine learning models directly on the device, without needing to send data to a centralized server for training.

B. DEEP LSTM AUTOENCODER LEARNING

An autoencoder is a category of neural network that can be trained to compress and reconstruct input data. It includes two networks: an encoder network that converts the input data into a compressed representation, and a decoder network that converts the compressed representation back into the original input data [25]. The performance of the model is measured by the ability to reconstruct the original data and is usually called reconstruction error. It is trained to reduce the difference between the input data and the reconstructed data, it can effectively grasp the structure and patterns in the data. A long short-term memory (LSTM) unit belongs to the class of recurrent neural networks (RNNs) that can represent long-term dependencies in time-series data [28]. The LSTM has been proposed to address the vanishing gradient problem which was the challenge of vanilla RNNs. The vanishing gradient problem can hinder the network's ability to grasp long-term dependencies in the data. The LSTM units have gates that regulate the flow of information and gradients, which allows the network to learn which parts of the input sequence are relevant for compression and which parts are not.

An LSTM autoencoder merges the capabilities of LSTM units and autoencoders to acquire a compressed representation of sequential data analysis. The LSTM units in the encoder and decoder are designed to handle time-series data and are able to model long-term dependencies. The input sequence is given as input to the LSTM encoder, which is analyzed and outputs a compressed representation of the data. The compressed representation is then fed into an LSTM decoder, which processes the data and reconstructs the original input sequence. The equation of LSTM gates and states, that are used for encoding and decoding is presented in Equation (1). During training, the parameters of the LSTM autoencoder are learned by minimizing the reconstruction error between the input data and the reconstructed output. The LSTM autoencoder compressed representation learned by the network can be applied for various tasks, such as sequence generation, anomaly detection, data compression, and prediction [29]. Although LSTM autoencoders are widely applicable it requires numerous amounts of resource for training.

$$\begin{aligned}
i_t &= \sigma\left(W_i . [h_{t-1}, x_t] + b_i\right)\\
f_t &= \sigma\left(W_f . [h_{t-1}, x_t] + b_f\right)\\
\tilde{C}_t &= tanh\left(W_c . [h_{t-1}, x_t] + b_c\right)\\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t\\
o_t &= \sigma\left(W_o . [h_{t-1}, x_t] + b_o\right)\\
h_t &= o_t * tanh(C_t)
\end{aligned} \tag{1}$$

where, $\mathrm{h_t}$ = hidden state, $\mathrm{o_t}$ = output gate, $\mathrm{C_t}$ = cell state, $\mathrm{i_t}$ = input gate, $\mathrm{f_t}$ = forget gate

C. RELATED WORK

Several studies have been undertaken to enhance the performance of incremental learning. Authors in [15], proposed an LSTM-based incremental learning approach by creating batches of streams. They resample the available dataset in six sequences of batches (sub-datasets) as required by the incremental learning. The parameters of LSTM (i.e., weights, biases, hidden and cell states) in the current are stored in memory for the next batch training instead of storing the historical data. However, the mathematical aspect of the proposed LSTM is not clearly defined and they didn't mention how the evaluation metrics are computed as knowledge is transferred from the previous batches. Authors in [22] introduce a typical incremental method for IIoT which gradually adjusts for varying domains and neither storing masks nor task labels are required. They have presented the performance using accuracy, forward and backward knowledge transfer, and parameters used, but they didn't recommend the specific neural network and how the data has been prepared for demonstrating the incremental nature. Laura Melgar-García [9] present a real-time prediction method for detecting anomalies and novelties using initial offline training followed by a real-time incremental approach. It employs nearest-neighbor techniques and incorporates novel elements in order to identify appropriate anomalies from real-time sequential data. Even though, offline training is essential real-world applications might not have appropriate baseline data.

Dai et al. [24] introduced a modified LSTM model called Hidden Layer LSTM (H-LSTM), where hidden layers are added and uses grow-and-prune to reduce run-time latency and number of parameters. They have applied magnitude-based pruning and gradient-based growth iteratively in order to adjust architecture of the network. This approach uses both pruning ang growth mechanisms at the same time in order to get the best fitting LSTM architecture, but it doesn't clearly prove the cost of these to algorithms. Li et al. [26] proposes threshold-based network pruning and nearest neighbor principle using the incremental learning approach for reducing the overhead of complex models. It uses an automatic weight masking method using the dispersion coefficients. It completely ignores old data and applies mean nearest neighbor for finding the set of fixed weights which makes all fixed weights to lower value and may bias the prediction. Yang et al. [13] proposes an adaptive network updated using class incremental approach for producing sparse indicator vectors and minimizing the issue of overfitting and catastrophic forgetting. To prevent confusion between classes, it dynamically samples feature vectors from class distributions and facilitates feature distribution. It speeds up feature distribution and dynamically samples feature vectors from class distributions to prevent confusion between classes. Although this approach brings a new insight, network expansion and old class distribution are extra resource consuming procedures before the actual incremental proceeds which is a bottle neck tiny device.

In summary, the aforementioned incremental learning methods have made significant strides by transferring parameters, and implementing network pruning and growing techniques. Specifically, authors in [13], [23], and [26] consider weight and network pruning (sparsening) for optimizing incremental learning. However, none of them considers applying weight sparsening together with taking samples of previously processed sub-dataset, which is the motivation of this study.

III. METHODOLOGY

Recent studies have proved that deep neural network training involves a high degree of redundancy of information [24], [26]. The same information is represented within the network, which can lead to inefficient resource utilization. This paper introduces an optimized incremental deep LSTM-autoencoder network for resource constraint devices of industrial applications. The dataset collected over time can be processed with a sequence of tasks. Hence, the proposed algorithm trains each task independently. This paper aims to minimize the resource wasted for processing redundant knowledge by dynamically sparsening the network weight matrix and sampling old data for the current task training.

The level of information redundancy could be determined by data variation [26]. This study utilizes the inverse of the coefficient of variation which is a statistical method that measures the variability of the dataset. The inverse of the coefficient of variation is computed for the network weights.

A higher value of the inverse of the coefficient of variation indicates the network weights are close to each other (closer to the mean) which suggests the higher redundancy of information. On the contrary lower value of the inverse of the coefficient of variation shows the network weights are spread out (far away from the mean) and suggests less redundancy of information. Hence, the network weight can be sparsened on each task training dynamically based on the inverse of the coefficient of variation. The inverse of the coefficient variation could be computed using mean and standard deviation.

The mean and standard deviation of weights on each task could be computed as

$$\mu_{task} = \frac{\sum_{i=0}^{N_{task}} w_i}{N_{task}} \tag{2}$$

$$\mu_{task} = \frac{\sum_{i=0}^{N_{task}} w_i}{N_{task}} \tag{3}$$

where, μ_{task} mean for the sub-dataset, σ_{task} standard deviation for the sub-dataset w_i = the set of weights sub-dataset D_i, N_{task} = the size of the weights for the sub-dataset

The coefficient of variation for weights is computed as,

$$w\_spar = ICV = a\frac{1}{CV} = a\frac{\mu_{task}}{\sigma_{task}} \tag{4}$$

where, CV = coefficient of variation, ICV = Inverse of the coefficient of variation, w_spar is ratio of weights to be sparsened and α is a regularizing parameter

The coefficient of variation determines the ratio of weights that can be sparsened on each training task, as shown in equation (4). The most important point that should be addressed is how to fairly distribute these percentages of weights throughout the weight matrix. In this case, we allocate for each column proportionally using the inverse of the coefficient of variation for each column and distribute it randomly. The relative proportion of the inverse of coefficient of variation computed for each column is used to distribute the weight sparsification. If a column has a higher inverse of the coefficient of variation, then it gets a higher proportion and gets sparser, and vice versa. The proportion of weights assigned for each column is distributed fairly above and below the mean. This way minimizes the performance loss caused by biased sparsification since it is evenly distributed across the weight matrix. The proposed approach has been articulated through pseudocodes (Algorithm 1 and Algorithm 2) which is language independent high-level description of a computer program.

A typical random matrix sparsening is depicted in Fig. 1. and the details of the process are as follows and also formally explained in Algorithm 1.

Step 1: Compute the inverse of the coefficient of variation for each column, equation (5).

$$ICV_{col_j} = \frac{\mu_{col_j}}{\sigma_{col_j}} \tag{5}$$

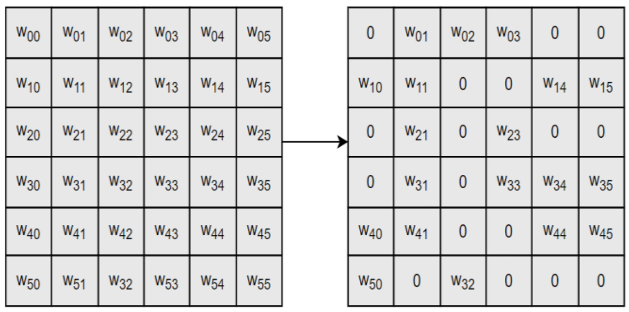

FIGURE 1. A typical weight matrix sparsening.

where, ICV_{col_j} is the inverse of the coefficient of variation for each column

Step 2: Find the relative proportion (Equation (6)) of each column based on the values of inverse coefficient of variation obtained in step 1.

$$rp\_col_j = \frac{ICV_{col_j}}{\sum_j ICV_{col_j}} \tag{6}$$

where, rp_col_j relative proportion of inverse of the coefficient of variation for each column

Step 3: Distribute the sparsened ratio of the weights obtained in Equation (4) to each column based on the relative proportion, (Equation (7)).

$$col\_spar_j = rp\_col_j * w\_spar \tag{7}$$

where, col_spar_j ratio of weights to be sparsened on each column.

Step 4: Compute the ratio of data points above and below the mean for each column and distribute the ratio of column sparse (col_sparj). Equations (8) and (9) compute the sparsened ratio for the column values above the mean, while the remaining ratio pertains to values below the mean.

$$ratio_{above\_mean} = \frac{N_{col_{above\_mean}}}{N_{col}} \tag{8}$$

$$col\_spar_{above\_mean} = ratio_{above\_mean} * col\_spar_j \tag{9}$$

Step 5: Find the number of data points to be sparsened above and below the mean for each column and distribute it randomly

This method of distribution reduces bias and improves performance since it traces almost the whole part of the weight matrix.

Training the entire older dataset or ignoring it brings its own trade-off as mentioned in the introduction part of this paper. Hence, a dynamic sampling mechanism for the group of previous datasets has been proposed. This mechanism allows to select a subset of the old dataset that is most relevant to the new task, which can improve performance while reducing computational cost. Suppose the dataset length 'n' is sufficient for training. In this case, the current sub-dataset is used as is, and the rest is obtained from previous sub-datasets. We weight the previous sub-datasets based on their

Algorithm 1 Weight Sparsification Algorithm

```
**Input**: sub-dataset $D_{sub}$, weight $w_{sub}$
**Output**: Sparsened weight
**while** training sub-dataset $D_{sub}$:
    (x,y) $\leftarrow$ set of sub dataset $D_{sub}$
    $W_{sub}$ $\leftarrow$ access the weights from the current training
    //Compute mean and standard deviation $\mu$ and $\sigma$
    w_spar $\leftarrow$ ICV $\leftarrow$ $\mu, \sigma$, a //compute ICV of
    $ICV_{col_j} \leftarrow \mu_{col_j}, \sigma_{col_j}$ //compute ICV for each column
    $col\_spar_j \leftarrow rp\_col_j$, w_spa
    $N_{col_{above\_mean}} \leftarrow col\_spar_j$
    $ratio_{above\_mean} \leftarrow N_{col_{above\_mean}}, N_{col}$
    $col\_spar_{above\_mean}, col\_spar_{below\_mean} \leftarrow ratio_{above\_mean}, col\_spar_j$
    //randomly sparse each column using the specific ratio
    $w_{sub\_spar} \leftarrow col\_spar_{above\_mean}, col\_spar_{below\_mean}$
    $w_{sub} \leftarrow w_{sub\_spar}$ //set sparsened weight for the next training
**endwhile**
```

Algorithm 2 Dynamic Sampling Mechanism

```
**Input**: sub-dataset $D_{sub}$
**Output**: Sampled sub-datasets
**while** training dataset:
    $CV_{subDi} \leftarrow \sigma_{subDi}, \mu_{subDi}$
    $rp\_subD_i \leftarrow CV_{subDi}, CV_{subDi}$
    $n_{subD_i} \leftarrow rp_{subDi}$, N,$no\_subD_N$
    $sam_{data} \leftarrow subD_N \cup n_{subD_1} \cup n_{subD_2} n_{subD_3} \ldots . \cup n_{subD_{N-1}}$
**Endwhile**
```

coefficient of variation to obtain the remaining part of the dataset (Equation (12)). This ensures that the most informative sub-datasets are given more weight, which can improve the performance of our model. If a certain sub-dataset has a higher coefficient of variation, then a higher amount of sample is taken from it and vice versa. The overall process is depicted in Algorithm 2. It is also important to set maximum and minimum proportions in order to minimize bias caused by taking extremely high or low amounts of data from a certain sub-dataset.

$$CV_{subDi} = \frac{\sigma_{subDi}}{\mu_{subDi}} \tag{10}$$

$$rp\_subD_i = \frac{CV_{subDi}}{\sum_{i=1}^{N-1} CV_{subDi}} \tag{11}$$

$$n_{subD_i} = rp_{subDi} * (N - no_{subDN}) \tag{12}$$

$$sam_{data} = subD_N \cup n_{subD_1} \cup n_{subD_2} n_{subD_3} \ldots . \cup n_{subD_{N-1}} \tag{13}$$

where, CV_{subDi} is coefficient of variation, μ_{subDi} is mean, σ_{subDi} standard deviation, n_{subD_i} number of samples for individual sub-dataset, no_subD_N number of the current sub-dataset, sam_{data} the total amount of sample and N is total amount of sample.

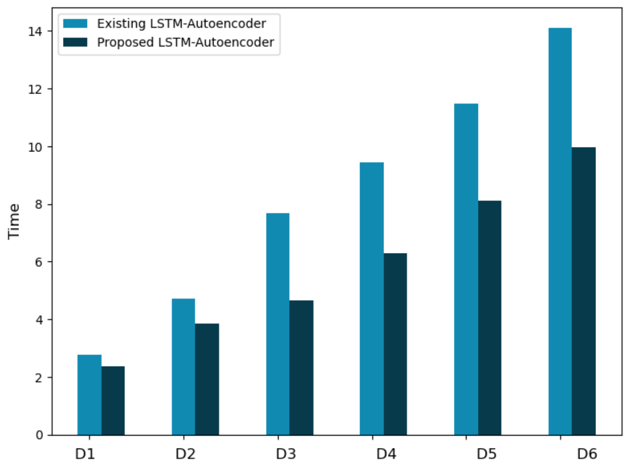

FIGURE 2. Comparison of the proposed and existing model on processing time using the weather dataset.

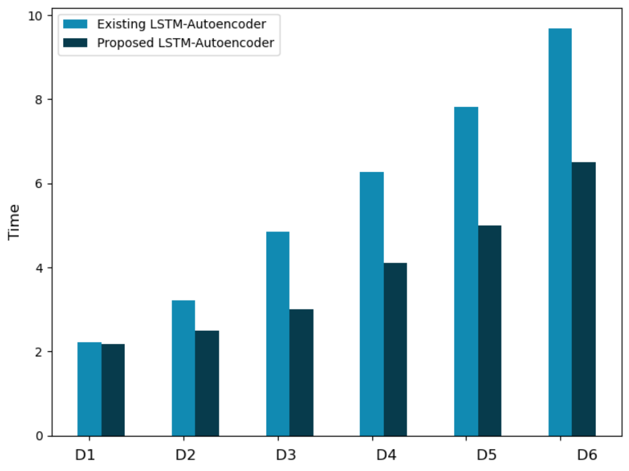

FIGURE 3. Comparison of the proposed and existing model on processing time using the thermostat dataset.

IV. RESULT AND DISCUSSION

The evaluation of the proposed approach has been undertaken using a testbed dataset collected at the University of New South Wales website [8]. The dataset was specifically prepared for testing security analysis of IoT and IIoT devices such as intrusion detection systems and anomaly detection. The measures of garage door, Modbus, smart fridge, smart motion sensor, Global Positioning System (GPS), weather monitoring system, and the smart thermostat has been reported. Among those, weather monitoring systems and thermostats are widely applicable in smart manufacturing [20]. Hence, we have chosen them for evaluating the performance of the proposed model. The experiment was conducted using Python with TensorFlow framework on a hardware setup consisting of an Intel (R) Core (TM) i5-8350U CPU (1.70 GHz) and 8 GB of main memory.

The dataset selection, preparation and ways of feeding to the algorithm is presented as follows. Sequential models require batches of sequential inputs usually ordered by time [15] and the evaluation of this study has been performed in

TABLE 1. Parameter configuration.

Parameter	Value
Loss function	Mean Square Error
Optimizer	Adam
Batch size	32
Epochs	20

TABLE 2. Results of BIC and AIC for both the proposed and existing methods.

Sub-dataset	Existing LSTM-AutoEn	Proposed LSTM-AutoEn	Existing LSTM-AutoEn	Proposed LSTM-AutoEn
	BIC for Weather		AIC for Weather	
D1	30771.02	30625.47	20885.27	20739.73
D2	49671.83	34505.46	38831.61	24368.58
D3	68135.73	42069.80	56737.36	31536.84
D4	86438.96	53270.67	74644.65	42299.23
D5	104706.46	67210.96	92605.02	55835.74
D6	122872.37	81806.76	110520.32	70246.67
	BIC for Thermostat		AIC for Thermostat	
D1	24484.42	24452.73	15129.74	15098.06
D2	37613.51	27006.72	27304.03	17400.79
D3	50339.68	32269.64	39472.00	22267.51
D4	63021.79	40172.98	51758.13	29732.34
D5	75463.96	49776.94	63893.13	38932.42
D6	87901.26	59658.61	76079.50	48483.76

order of phases. As the incremental learning is a modular approach which requires the use of sub-datasets for the subsequent phase of training [14], [15], we subdivide the available dataset into six groups chronologically. Thus, the groups of the dataset represent real-time batches of data that arrived at different time sequences and are labeled as D1, D2, D3, D4, D5, and D6. Therefore, both the weather monitoring system dataset and the smart thermostat dataset, which are used in this experiment, have been grouped into six sub-datasets. The ways of feeding the subgroup of datasets to the algorithm is not the same for the proposed and existing methods. The existing method which is used as a baseline applies direct concatenation of the available sub-datasets for the current phase of training. However, the proposed method takes sample of the former sub-datasets during the instant training phases. Table (1) presents the experimental settings for the proposed model.

The processing time, accuracy, precision, recall, information criteria (Akaike Information Criteria (AIC) and Bayesian Information Criteria (BIC)) have been used for comparative evaluation [2], [30]. Accuracy, precision, and recall are common metrics for evaluating the performance of a classification model and processing time has been applied for evaluating the resource utilization. Accuracy measures the model's ability to predict correct classifications. Precision is the proportion of positive predictions that are actually correct whereas recall measures the model's ability to correctly identify existing positives. We also undertake a comparison using AIC and BIC for measuring the complexity of the model and how the model fits the data.

Fig. 2. illustrates the processing time of the six sub-datasets for both existing and proposed methods. The proposed approach shows a significant result from the second sub-dataset. The reason behind this significant change is due to the proposed approach's systematic sampling technique in order not to feed the whole available dataset at the given phase of training. Table (3) and (4) shows the accuracy, precision, and recall of both the existing and proposed approaches. As clearly shown in both tables, there is no significant change observed in the existing and proposed approach. Table (2) presents the results of a comparative evaluation for the proposed and existing approaches using AIC and BIC metrics. The proposed approach achieves lower AIC and BIC values compared to the existing approach. With this, we prove that the proposed method would improve the resource required for training without losing prediction performance.

TABLE 3. Performance comparison on weather dataset.

Sub-dataset	Accuracy		Precision		Recall	
	Existing LSTM-AutoEn	Proposed LSTM-AutoEn	Existing LSTM-AutoEn	Proposed LSTM-AutoEn	Existing LSTM-AutoEn	Proposed LSTM-AutoEn
D1	65	69	59	61	59	62
D2	67	65	58	59	63	64
D3	67	70	69	68	66	68
D4	68	69	56	58	59	70
D5	70	73	67	70	74	78
D6	71	68	57	60	70	68

TABLE 4. Performance comparison on thermostat dataset.

Sub-dataset	Accuracy		Precision		Recall	
	Existing LSTM-AutoEn	Proposed LSTM-AutoEn	Existing LSTM-AutoEn	Proposed LSTM-AutoEn	Existing LSTM-AutoEn	Proposed LSTM-AutoEn
D1	64	63	57	61	59	62
D2	63	65	59	58	60	58
D3	68	67	65	66	64	68
D4	73	72	68	69	70	69
D5	67	65	62	60	68	62
D6	70	70	57	55	70	70

V. CONCLUSION

Industrial control systems require intelligent data processing mechanisms to be deployed closer to the edge of the network in order to improve their efficiency and minimize delays. Incremental learning is the most effective way to analyze industrial data at the edge, but it requires careful resource optimization. This study presented resource efficient LSTM-autoencoder model using statistical dispersion in order to sparse the network weights and take samples of previously processed data for resource-deficient edge devices. The experimental demonstration with two IIoT-related testbed datasets proved the effectiveness of the proposed approach by reducing computational time without compromising performance. Such a lightweight approach enables real-time data processing and decision-making which is a crucial for industrial control applications.

The amount of data increases through time which needs significant amount of resource for data analytics and storage space. Hence, excluding older sub-datasets from data sampling could be a future direction of this study, as it saves resources without harming prediction performance. Furthermore, the experiment can be extended by incorporating data streams with real edge devices using different programming languages and platforms.

REFERENCES

[1] H. Benaddi, M. Jouhari, K. Ibrahimi, J. Ben Othman, and E. M. Amhoud, "Anomaly detection in industrial IoT using distributional reinforcement learning and generative adversarial networks," *Sensors*, vol. 22, no. 21, p. 8085, Oct. 2022, doi: 10.3390/s22218085.

[2] A. K. Takele and B. Villányi, "Resource aware long short-term memory model (RALSTMM) based on-device incremental learning for industrial Internet of Things," *IEEE Access*, vol. 11, pp. 63107–63115, 2023, doi: 10.1109/ACCESS.2023.3289076.

[3] X. Wang, S. Garg, H. Lin, J. Hu, G. Kaddoum, M. J. Piran, and M. S. Hossain, "Toward accurate anomaly detection in industrial Internet of Things using hierarchical federated learning," *IEEE Internet Things J.*, vol. 9, no. 10, pp. 7110–7119, May 2022, doi: 10.1109/JIOT.2021.3074382.

[4] A. K. Takele and B. Villányi, "Anomaly detection using hybrid learning for industrial IoT," in *Proc. IEEE 2nd Conf. Inf. Technol. Data Sci. (CITDS)*, Debrecen, Hungary, May 2022, pp. 262–266, doi: 10.1109/CITDS54976.2022.9914338.

[5] W. Wu, L. Shen, Z. Zhao, M. Li, and G. Q. Huang, "Industrial IoT and long short-term memory network-enabled genetic indoor-tracking for factory logistics," *IEEE Trans. Ind. Informat.*, vol. 18, no. 11, pp. 7537–7548, Nov. 2022, doi: 10.1109/TII.2022.3146598.

[6] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "Fed-IIoT: A robust federated malware detection architecture in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 17, no. 12, pp. 8442–8452, Dec. 2021, doi: 10.1109/TII.2020.3043458.

[7] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "ICaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 5533–5542, doi: 10.1109/CVPR.2017.587.

[8] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems," *IEEE Access*, vol. 8, pp. 165130–165150, 2020, doi: 10.1109/ACCESS.2020.3022862.

[9] L. Melgar-García, D. Gutiérrez-Avilés, C. Rubio-Escudero, and A. Troncoso, "Identifying novelties and anomalies for incremental learning in streaming time series forecasting," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106326, doi: 10.1016/j.engappai.2023.106326.

[10] A. Kulshrestha, L. Chang, and A. Stein, "Use of LSTM for sinkhole-related anomaly detection and classification of InSAR deformation time series," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 15, pp. 4559–4570, 2022, doi: 10.1109/JSTARS.2022.3180994.

[11] S. Wang, P. Lin, R. Hu, H. Wang, J. He, Q. Huang, and S. Chang, "Acceleration of LSTM with structured pruning method on FPGA," *IEEE Access*, vol. 7, pp. 62930–62937, 2019, doi: 10.1109/ACCESS.2019.2917312.

[12] P. Narkhede, R. Walambe, S. Poddar, and K. Kotecha, "Incremental learning of LSTM framework for sensor fusion in attitude estimation," *PeerJ Comput. Sci.*, vol. 7, p. e662, Aug. 2021, doi: 10.7717/peerj-cs.662.

[13] B. Yang, M. Lin, Y. Zhang, B. Liu, X. Liang, R. Ji, and Q. Ye, "Dynamic support network for few-shot class incremental learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 2945–2951, Mar. 2023, doi: 10.1109/TPAMI.2022.3175849.

[14] H. Wang, M. Li, and X. Yue, "IncLSTM: Incremental ensemble LSTM model towards time series data," *Comput. Electr. Eng.*, vol. 92, Jun. 2021, Art. no. 107156, doi: 10.1016/j.compeleceng.2021.107156.

[15] Á. C. L. Neto, R. A. Coelho, and C. L. D. Castro, "An incremental learning approach using long short-term memory neural networks," *J. Control, Autom. Electr. Syst.*, vol. 33, no. 5, pp. 1457–1465, Oct. 2022, doi: 10.1007/s40313-021-00882-y.

[16] D. Kong, X. Li, and W. Chen, "Automatic model adaption method based on few-shot incremental learning for IoT applications," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Foshan, China, Aug. 2022, pp. 417–422, doi: 10.1109/ICCC55456.2022.9880763.

[17] H. R. Tavakoli, J. Wabnig, F. Cricri, H. Zhang, E. Aksu, and I. Saniee, "Hybrid pruning and sparsification," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Anchorage, AK, USA, Sep. 2021, pp. 3542–3546, doi: 10.1109/ICIP42928.2021.9506632.

[18] H. T. Truong, B. P. Ta, Q. A. Le, D. M. Nguyen, C. T. Le, H. X. Nguyen, H. T. Do, H. T. Nguyen, and K. P. Tran, "Light-weight federated learning-based anomaly detection for time-series data in industrial control systems," *Comput. Ind.*, vol. 140, Sep. 2022, Art. no. 103692, doi: 10.1016/j.compind.2022.103692.

[19] H. Liu, Y. Zhou, B. Liu, J. Zhao, R. Yao, and Z. Shao, "Incremental learning with neural networks for computer vision: A survey," *Artif. Intell. Rev.*, vol. 56, no. 5, pp. 4557–4589, May 2023, doi: 10.1007/s10462-022-10294-2.

[20] X. Yu, X. Yang, Q. Tan, C. Shan, and Z. Lv, "An edge computing based anomaly detection method in IoT industrial sustainability," *Appl. Soft Comput.*, vol. 128, Oct. 2022, Art. no. 109486, doi: 10.1016/j.asoc.2022.109486.

[21] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, "Toward edge-based deep learning in industrial Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4329–4341, May 2020, doi: 10.1109/JIOT.2019.2963635.

[22] Y. Zhao, D. Saxena, and J. Cao, "Memory-efficient domain incremental learning for Internet of Things," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, Nov. 2022, pp. 1175–1181, doi: 10.1145/3560905.3568436.

[23] S. Dang, Z. Cao, Z. Cui, Y. Pi, and N. Liu, "Class boundary exemplar selection based incremental learning for automatic target recognition," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 8, pp. 5782–5792, Aug. 2020, doi: 10.1109/TGRS.2020.2970076.

[24] X. Dai, H. Yin, and N. K. Jha, "Grow and prune compact, fast, and accurate LSTMs," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 441–452, Mar. 2020, doi: 10.1109/TC.2019.2954495.

[25] W. Yu, Y. Liu, T. Dillon, and W. Rahayu, "Edge computing-assisted IoT framework with an autoencoder for fault detection in manufacturing predictive maintenance," *IEEE Trans. Ind. Informat.*, vol. 19, no. 4, pp. 5701–5710, Apr. 2023, doi: 10.1109/TII.2022.3178732.

[26] X. Li, S. Dong, Q. Su, M. Yu, and X. Li, "Adaptive threshold hierarchical incremental learning method," *IEEE Access*, vol. 11, pp. 12285–12293, 2023, doi: 10.1109/ACCESS.2023.3242688.

[27] I. Priyadarshini, R. Kumar, A. Alkhayyat, R. Sharma, K. Yadav, L. M. Alkwai, and S. Kumar, "Survivability of industrial Internet of Things using machine learning and smart contracts," *Comput. Electr. Eng.*, vol. 107, Apr. 2023, Art. no. 108617, doi: 10.1016/j.compeleceng.2023.108617.

[28] C. Zhang, H. Pang, J. Liu, S. Tang, R. Zhang, D. Wang, and L. Sun, "Toward edge-assisted video content intelligent caching with long short-term memory learning," *IEEE Access*, vol. 7, pp. 152832–152846, 2019, doi: 10.1109/ACCESS.2019.2947067.

[29] H. Zhu, S. Liu, and F. Jiang, "Adversarial training of LSTM-ED based anomaly detection for complex time-series in cyber-physical-social systems," *Pattern Recognit. Lett.*, vol. 164, pp. 132–139, Dec. 2022, doi: 10.1016/j.patrec.2022.10.017.

[30] S. Chaturvedi, E. Rajasekar, S. Natarajan, and N. McCullen, "A comparative assessment of SARIMA, LSTM RNN and fb prophet models to forecast total and peak monthly energy demand for India," *Energy Policy*, vol. 168, Sep. 2022, Art. no. 113097, doi: 10.1016/j.enpol.2022.113097.

ATALLO KASSAW TAKELE received the M.Sc. degree in computer science (computer networking stream) from Jimma University, Jimma, Ethiopia, in 2018. He is currently pursuing the Ph.D. degree in computer engineering with the Budapest University of Technology and Economics, Budapest, Hungary. His research interests include machine and deep learning, the IoT, Industry 4.0, security, and enterprise application integration.

BALÁZS VILLÁNYI received the Ph.D. degree in computer science from the Budapest University of Technology and Economics, Budapest, Hungary. He is currently an Associate Professor and a Doctoral Advisor with the Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics. His research interests include machine learning, schema matching algorithms, enterprise application integration, the Industrial IoT, and Industry 4.0.