A Novel Bearing Fault Diagnosis Method based on Stacked Autoencoder and End-edge Collaboration

Chen Yang
School of Cyberspace Science and Technology
Beijing Institute of Technology
Beijing, P.R.China
yangchen666@bit.edu.cn

Zou Lai
School of Computer Science and Technology
Beijing Institute of Technology
Beijing, P.R.China
laizou1002@163.com

Yingchao Wang
School of Cyberspace Science and Technology
Beijing Institute of Technology
Beijing, P.R.China
wyc_mail@163.com

Shulin Lan*
School of Economics and Management
University of Chinese Academy of Science
Beijing, P.R.China
Corresponding author: lanshulin@ucas.ac.cn

Lihui Wang
Department of Production Engineering
KTH Royal Institute of Technology
Stockholm, Sweden
lihuiw@kth.se

Liehuang Zhu
School of Cyberspace Science and Technology
Beijing Institute of Technology
Beijing, P.R.China
liehuangz@bit.edu.cn

***Abstract*—The deep learning based fault diagnosis methods show excellent performance. However, cost and delay factors make it difficult for their widespread industrial application. Microcontroller units (MCUs) in industrial equipment have the advantages of real-time response and high reliability and usually have some redundant computational resource. However, even lightweight deep learning models cannot be deployed in MCUs due to severely limited computational resources. This paper proposes an end-edge collaborative fault diagnosis framework, by combining real-time decision-making at the end with dynamic adaptive diagnosis at the edge to improve inference performance. The model's minimum input size is deduced through theoretical analysis of the bearing working mechanism, and to make the model suitable for MCUs, we leverage the differential characteristics of the bearing vibration data and proposed a TinyML model based on stacked autoencoders. The pre-autoencoder extracts differential features, while the post-autoencoder performs fault diagnosis based on pooled differential features. Finally, the stacked-autoencoder model and collaborative framework were evaluated using the CWRU bearing dataset, achieving 384x compression in parameter size and 100% accuracy for binary fault classification, requiring only 6.44kB RAM. With the dynamic adaptive collaboration mechanism, the proposed fault diagnosis framework can reduce the edge load by approximately 94%.**

***Index Terms*—Fault diagnosis, end-edge collaboration, TinyML, deep learning, stacked autoencoder, differential feature**

I. Introduction

Rolling bearings, a key component of rotating machinery, play an extremely important role in modern industries such as wind power generation and rail transportation, where bearing failures have been shown to be a major cause of machine breakdown and even more serious accidents [1]. Therefore, there is an urgent need for continuous monitoring and highly accurate fault diagnosis of bearings in high-risk or high-value equipment.

Deep learning has been successfully applied in the field of fault diagnosis because of its powerful feature representation ability, however, there still exist problems:

1) *The contradiction between inference accuracy and latency:* Existing research usually employs large-scale deep learning models to achieve high inference accuracy. The cloud/edge based fault diagnosis models can achieve high accuracy but incur large delays and are not suitable for time-critical fault detection applications.
2) *Ineffective utilization of the physical characteristics of faults:* Repetitive pulses are typical features of faulty bearings [2], but existing studies usually try to improve model performance by adjusting the neural network, without considering important periodic physical features caused by bearing faults.
3) *Difficulty in using computational resources of massive end devices:* Shop floors have massive control chips such as the microcontroller unit (MCU) and digital signal processor in industrial facility, but it is difficult to run deep learning models or even lightweight ones in an end device because of severely limited computational resources.

The tiny machine learning (TinyML) based inference can be accomplished in the MCU, which brings great advantages in terms of timeliness, cost and privacy [3]. In this paper, we proposed a novel fault diagnosis method based on TinyML and end-edge collaboration to exploit massive and distributed end computing resource, and to adaptively use edge computing for collaborative fault diagnosis with low latency and high accuracy. Our contributions include:

1) By designing the TinyML diagnosis model at the end, computational resource of massive end devices can be effectively utilized, and the response time is significantly reduced. Edge computing is adaptively used to conduct inference with bigger models to increase the diagnosis accuracy.
2) Combining the physical vibration characteristics of faulty bearings, the minimum required number of input nodes (sampling points) for the fault diagnosis model is theoretically determined, and the differential vibration characteristics are analyzed to provide theoretical support for accurate diagnosis.
3) Considering extremely limited end computing resource, a bearing fault diagnosis model based on stacked autoencoders is designed for the MCU, and the peak RAM occupation is dramatically reduced through network struc-

ture optimization. It realizes effective usage of remaining resource of massive end devices for real-time prediction.

The rest of this paper is organized as follows: Section II introduces related work; Section III proposes a fault diagnosis framework with collaborative edge-end processing; Section IV analyzes the number of sampling points required and differential characteristics of bearing vibration data and proposes a fault diagnosis model based on stacked autoencoders; Section V verifies the model and the collaborative framework in terms of accuracy and time delay based on the Case Western Reserve University (CWRU) dataset; Section VI concludes the paper with final remarks.

II. Related Work

A. Tiny machine learning (TinyML)

Lin et al [4] proposed a framework for jointly designing efficient neural architecture (TinyNAS) and lightweight inference engine (TinyEngine) to implement ImageNet-scale inference on microcontrollers. Liu et al [5] replaced each large dense layer of DNN with three small cascaded sub-layers, therefore replacing the computation/storage of a large matrix with that of small ones, and designed a new rank-restricted back-propagation algorithm to enable the deployment of tiny AI models onto low-cost and low-power devices. An autoencoder with an additional layer for online fine-tuning is proposed in [6]. A common model is first trained and after model deployment, the end device performs incremental learning based on the stream data collected in real time to deal with the concept drift.

Scholars have conducted intensive research on neural network optimization and compression to minimize the resource consumption of the TinyML model. However, current optimization methods mostly involve neural network improvement and negelect to incorporate the useful physical characteristics of faulty rolling bearing.

B. Edge-cloud collaborative fault diagnosis

Yang et al [7] proposed a cloud-edge-end collaboration framework for cloud manufacturing, and analyzed the deployment and update mechanism of deep learning models to effectively support fast response and high-performance decision-making for industrial robots. Shao et al [8] proposed Branchy-GNN for efficient graphical neural network (GNN)-based point cloud processing by leveraging edge computing and branch networks with early exiting to significantly reduce communication overhead as well as latency. Fang et al [9] proposed an on-demand DNN model inference system for industrial edge nodes with knowledge distillation and early exit, which improves inference latency and memory usage by training compact edge models under the supervision of large complex models and using an early exit mechanism in the inference phase.

Current studies mainly focus on the collaboration mechanisms between the cloud, edge and smart end devices, and does not deal with effective utilization of massive resource-limited MCUs in industrial systems.

III. End-edge collaborative fault diagnosis framework

A. Fault diagnosis framework based on end-edge collaboration

We propose an end-edge collaborative fault diagnosis framework, shown in Fig.1. TinyML models and a high performance model are deployed at the end devices and the edge node respectively. End-edge collaboration for fault diagnosis has two operation modes: end decision-making and edge decision-making. The TinyML based inference is first conducted at the end node for decision making of fault diagnosis. Whether to use the local inference result is judged by the dynamic constraint algorithm in Section III-B. The second mode is end-edge collaborative decision-making. When the end inference result does not satisfy the two conditions, the data is uploaded to the edge node, which performs inference and judges whether the two conditions are satisfied.

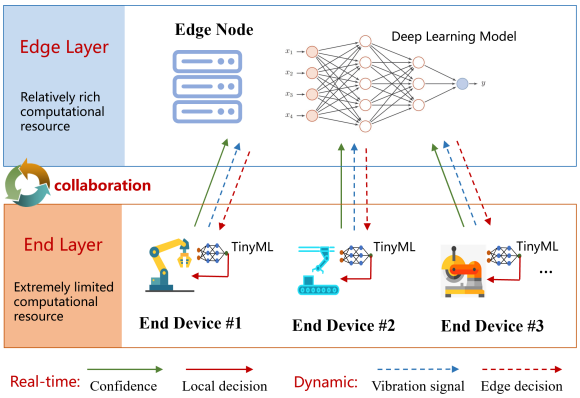


Fig. 1. End-edge collaborative fault diagnosis framework.

B. End-edge coordination mechanism

Industrial fault diagnosis demands accuracy and low delay. Edge computing alone can't ensure rapid inference due to calculation delay and network instability. TinyML models on end devices allow for rapid response but may lack precision. Thus, we propose an end-edge collaborative decision-making algorithm that considers both the constraints of delay and confidence.

After inference, the end node calculates output confidence

$$C(z) = \frac{e^{z}}{\sum_{j=1}^{K} e^{z_j}} \tag{1}$$

where z is the output of the neural network, j is the index of fault classification, and K is the number of fault classes.

To enable end-edge coordination, the edge node can utilize larger and more powerful deep learning models and conduct on-demand fault diagnosis to its subordinate end nodes in a dynamic and adaptive manner. The end device/equipment first calculates the confidence of each prediction. When the confidence of a prediction on an end node is lower than a threshold, the confidence is reported to the edge node, which will conduct inference instead according to the confidence level and send the results back, and vice versa. The edge node can perform more accurate inferences and identify specific fault categories (such as inner/outer race faults, ball faults, cage faults, etc.).

Algorithm 1 is deployed on each end node. d_e is the maximum allowable latency for the current fault diagnosis

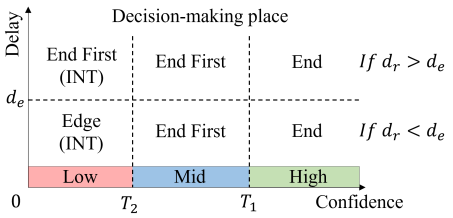


Fig. 2. End-edge dynamic collaborative decision-making.

task. The response time d_r of the edge node is determined by the delay of the end-edge bidirectional communication under the current network state and the inference time of the edge node. T_1 and T_2 indicate a standard threshold and a critical threshold of prediction confidence. If the inference confidence of an end node exceeds T_1, it indicates that the inference result is reliable. If the inference confidence falls within the range between T_1 and T_2, it implies that the inference result's reliability is somewhat diminished but still acceptable. However, if the inference confidence is lower than T_2, it indicates that the inference result is highly unreliable.

Algorithm 1: End work under delay and confidence constraints.

```
Execute the inference.
Calculate the confidence level C(z).
Obtain the response time d_r and constraint time d_e for this fault diagnosis task.
if d_r < d_e then
    if C(z) ≥ T_1 then
        Local decision and return.
    else if T_2 ≤ C(z) < T_1 then
        Local decision.
        Upload confidence to edge node, and return.
    else
        Issue an interrupt request to the edge.
    end
else
    if C(z) ≥ T_1 then
        Local decision and return.
    else if T_2 ≤ C(z) < T_1 then
        Local decision.
        Upload confidence to edge node, and return.
    else
        Local decision.
        Issue an interrupt request to the edge.
    end
end
```

Fig. 2 illustrates the collaborative decision-making process under the constraints of delay and confidence. The labels "End" and "Edge" in the figure signify that the end and edge models are employed for decision-making, respectively. The label "End First" implies that the inference result of the end node model is utilized for decision-making before obtaining the decision results from the edge node. This is due to the fact that even when the confidence level falls within the range between T_1 and T_2, the current end node's reliability may have slightly decreased, but it is still dependable overall. By the "End First" mechanism, the speed of end decision-making is combined with the reliability of edge decision-making, allowing end devices to make decisions rapidly. Even in rare instances where the prediction result of the end node is incorrect, this error can be promptly detected and rectified by the edge node. It should be noted that the suffix 'INT' in the label denotes an interrupt request. In scenarios where the inference confidence falls between the thresholds T_1 and T_2, intervention from the edge node remains necessary. However, the end node only transmits the confidence level instead of interrupt requests, to preserve the edge node's real time computing resources for more pressing and significant situations.

Algorithm 2: Real-time response of edge nodes.

```
while True do
    Record diagnostic requests from end nodes in the diagnosis queue Q.
    Refresh EC in real time while any end node uploads the inference confidence.
end
```

Algorithm 3: Dynamic diagnosis of edge nodes.

```
while True do
    while Q is non-empty do
        Sort Q in ascending order based on confidence level.
        Diagnose the first request in the queue Q.
        Send this diagnostic result to the end node.
        Mark the diagnosed node in Q as empty.
    end
    Diagnose an end node according to Arg min_{i=1,2...M}(EC).
    Send this diagnostic result to the end node.
    Mark the diagnosed node in EC as empty.
end
```

Algorithms 2 and 3 are executed at edge nodes by multi-threading. Algorithm 2 is used to respond to requests of end nodes in real time, including updating interrupt sequence $\boldsymbol{Q}$ and confidence set $\boldsymbol{EC}$ for end nodes. $\boldsymbol{EC} = \{C(z)\}$, where $C(z)$ is the inference confidence of the *z-th* end node. The edge server dynamically diagnoses the end nodes according to Algorithm 3. M is the number of end devices under this edge node. The interrupt mechanism in the edge node ensures that the emergency request of the end node can be responded immediately.

By utilizing Algorithm 1, 2, and 3, dynamic end-edge collaboration that satisfies confidence and delay constraints can be accomplished. This approach enables the efficient utilization of computing resources of end devices while reducing the workload on edge nodes.

IV. TinyML based fault diagnosis model

A. Model setting based on periodic fault characteristics

It is necessary to compress the fault diagnosis model as much as possible while ensuring the necessary performance to fit it into the highly resource-limited MCU. Current TinyML studies mostly tried to improve the model itself and does not effectively utilize the physical characteristics of bearing faults.

When the bearing works normally, the main frequency of the vibration is the inherent frequency of the bearing components, which shows good periodicity. For a faulty bearing, when mechanical contact occurs in the damaged position, a local impact force will be generated, forming pulse incentives and stimulating the vibration of bearing components. In other words, the vibration frequency of irregular shock events varies when different parts of the bearing (outer raceway, inner raceway or roller) fail. The characteristic frequency can be obtained from the analysis of motion relationship between the bearing components based on the rotational speed, the shape and the size of the bearing parts [10].

According to the parameters of the bearing used in the CWRU dataset, the characteristic frequencies of the driving end (DE) at the speed of 1797 rpm can be calculated as: $f_i = 162$Hz, $f_o = 107$Hz, $f_e = 11.93Hz$, and $f_b = 141$Hz. f_i, f_o, f_e, and f_b are ball pass frequency inner raceway, ball pass frequency outer raceway, fundamental train frequency and ball spin frequency respectively. The minimum difference of characteristic frequencies is $\Delta f_{\min} = f_i - f_b = 21Hz$.

From the sampling theorem for frequency domain, the frequency resolution can be caculated as

$$F_r = \frac{f_s}{N} \tag{2}$$

where f_s is the sampling frequency, and *N* is the number of sampling points in each cycle. For the CWRU dataset, f_s is 12 kHz. The frequency resolution needs to be no greater than the minimum difference of the characteristic frequencies, so

$$N = \frac{f_s}{F_r} \geq \frac{f_s}{\Delta f_{\min}} \approx 572 \tag{3}$$

Therefore, the number of sampling points in each cycle should not be smaller than 572, otherwise the resolution for frequency domain cannot support the TinyML model to accurately classify fault classses. It means that the input size of the autoencoder should not be less than 572.

B. Fault diagnosis model based on stacked autoencoder

Stacked autoencoders is a deep neural network created by using autoencoders as building blocks. Compared with traditional autoencoder, stacked autoencoders use the layer-by-layer pre-training and fine-tuning approach to improve the convergence speed and realize effective extraction of more abstract and complex fault feature.

For the neural network consisting of two cascaded autoencoders, its training is divided into two stages. In the first stage, the front autoencoder is trained with the original vibration data X under normal operating conditions, and the training objective is to minimize the reconstruction error e. In the second stage, the second autoencoder is trained with the results e_m from pooling operation of the front-autoencoder output under normal operating conditions, and the training goal is to minimize the reconstruction error of the input data e_m. Similarly, the training process of the *n* cascaded autoencoders is divided into *n* stages.

In order to further amplify the differential features, the output value of the previous model is maximally pooled.

$$\hat{e}(t) = \max_{i=t,t-1,\ldots,t-k} e(t) \tag{4}$$

where $e(t)$ is the Pre-MSE at moment *t*, $\hat{e}(t)$ is the Pre-MSE value at moment *t* after pooling, and k is the pooling depth.

For the network with two stacked autoencoders, n_1 consecutive samples are used as the input of the front autoencoder, and the MSE of this autoencoder the feature values $e(t)$. Each *k* consecutive feature values are pooled maximally to obtain $e_m(t)$, which is used as the n_2 input of the back autoencoder. The MSE of the back autoencoder is used to determine whether a fault occurs. A single differential feature is calculated from the n_1 original data, so after replacing the original vibration signal with differential features and pooling, the perceptual field of a single input node of the neural network is increased from 1 to $k \times n_1$. The MSE value at the output of the post network is determined by the continuous raw vibration data $n_1 \times n_2 \times k$.

For the CWRU dataset, taking the autoencoder with 3 hidden layers as an example, the number of input nodes is $ceil(\sqrt{572}) = 24$ if two autoencoders are cascaded and $ceil(\sqrt[3]{572}) = 9$ if three autoencoders are cascaded. Assume that the end device can calculate 6.72 MFLOPs and the RAM is 196kB (a common microcontroller). A further analysis of the number of cascaded autoencoders is shown in the table, taking a symmetric autoencoder with the number of neurons in each layer scaled down in turn to 2/3 that of the prior layer in the encoding stage and a pooling depth 4 as an example.

TABLE I
COMPARISON OF RESOURCE CONSUMPTION OF DIFFERENT MODELS

Cascade Depth	Input size	FLOPs	Parameters	RAM	Delay
1	572	631828	633419	2475kB	94.03ms
2	24	2359296	1648	6.44kB	351.09ms
3	9	60742656	543	2.13kB	9039.09ms

As can be seen from the Table I, the stacked-autoencoder network with two autoencoders reduces the RAM occupation by 99.74% through changing the space with time, compared to a traditional autoencoder. Although the inference time increases, it is still within an acceptable range. The network with 3 cascaded autoencoders results in further reduction in RAM occupation by 66.93%, but the amount of calculations shows a geometric increase, and the time of a single inference increases significantly from 351.09ms to 9039 ms. In summary, in this paper, a 2-autoencoder cascade model is used and the network structure is designed as shown in Fig. 3.

Through cascading, the receptive field [11] of the post autoencoder is increased, and the MSE of each pre-autoencoder output can indirectly reflect the features contained in 96

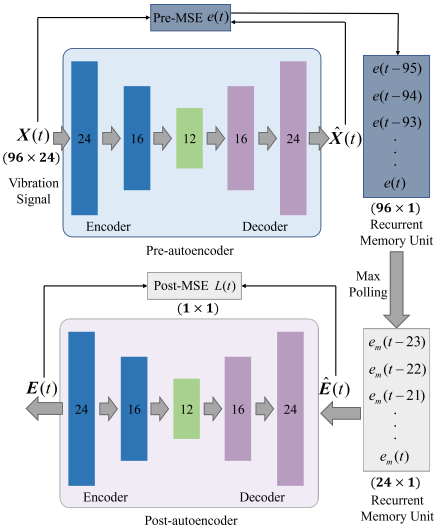


Fig. 3. Model structure with stacked autoencoders.

consecutive pieces of input data. Using two autoencoders, each with 24 input nodes, the MSE values of post-autoencoder output are determined by 2304 consecutive original data points $(24 \times 24 \times 4)$. Thus, the size of the input nodes is significantly reduced by using cascading. For TinyML, the biggest limitation in the inference stage for end hardware is the peak memory consumption, which is positively correlated with the number of input nodes, and the model depth does not significantly affect the peak memory consumption [4]. By cascading, the input size is greatly reduced, which in turn significantly reduces the peak memory occupation.

In the inference stage, during normal operating conditions, the pre-autoencoder can well reconstruct the vibration signal and the Pre-MSE can be stable at a smaller value, and the post-autoencoder can well reconstruct its input and the Post-MSE is also small. If a fault occurs, the distribution of the vibration signal changes, the output of the pre-autoencoder reflects larger differential characteristics, and the Pre-MSE will fluctuate greatly, and the post-autoencoder has difficulty to reconstruct its input, so the Post-MSE is larger. Therefore, accurate fault diagnosis can be performed by setting a reasonable threshold for Post-MSE.

It is worth noting that during the training process, the vibration signal after being sliced is shuffled. This means that only 24 consecutive samping points of the vibration signal need to be collected each time. After the pre-autoencoder reasoning ends, the memory can be released. This feature is of great significance for MCUs with severely limited memory resource.

V. Experiment

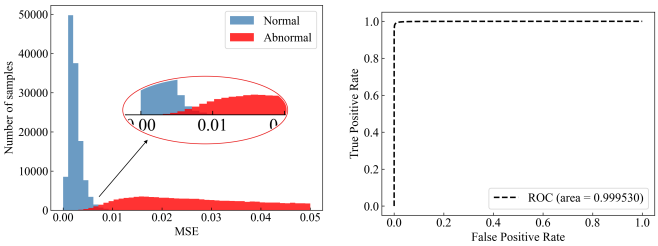
The proposed methodology was validated on the CWRU bearing dataset, which includes normal conditions and three types of failures (outer race, inner race, and ball) with three damage depths each. Signals were sampled at 12 kHz, 1800 rpm. 60% of the dataset was used to train the stacked autoencoders.

A. Fault diagnosis experiment using stacked autoencoders

Referring to the STM32F407 MCU commonly used in industry, with a clock frequency of 168MHz and 192 kB RAM, the redundant 20% of resources can be allocated to deploy a fault diagnosis algorithm. For the TinyML model, the MCU delivers about 6.72 MFLOPS computing capability and 38.4 kB RAM. Table I shows calculations and parameters of three stacked autoencoder models. The traditional model requires 2475 kB RAM, while typical MCUs only have a few hundred kB. Conversely, the stacked autoencoder converts space to time, reducing parameters to 6.44 kB, enabling deployment on common microcontrollers. The inference time of 351.09 ms meets bearing fault diagnosis requirements.

Further, the CWRU dataset is merged into a binary classification dataset. The merged data set includes two parts, which respectively contain healthy samples and fault samples in various situations (speed, load, fault location, fault level). The merged dataset is divided into samples (each with 24 points) for the experiments.

The Model structure shown in Fig. 3. The Pre-autoencoder and Post-autoencoder share same structure with 24, 16, 12, 16, and 24 neurons. The pooling depth is 4. We shuffle all samples before training, use the Adam optimization algorithm for training, and lr=0.001. The MSE is used as loss function.



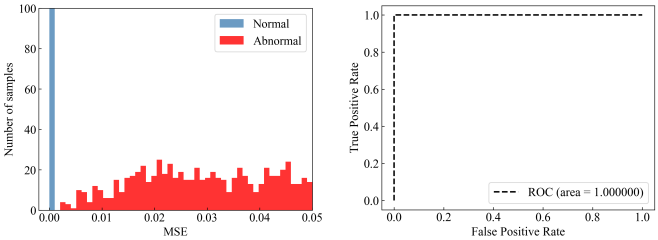
(a) Kernel density of MSE (b) ROC curve

Fig. 4. The MSE and ROC of traditional stacked autoencoders

TABLE II
THE RESULT OF TRADITIONAL STACKED AUTOENCODERS

Datasets	Healthy	Fault	FP	TN	Precision
Training set	101844	0	947	0	/
Test set	25461	265218	718	5304	98.00%

Fig. 4 is the MSE kernel density and ROC curve for traditional stacked autoencoders. Fig. 4(a) shows significant overlap between normal and abnormal samples, making it impossible to distinguish with fixed threshold. In contrast, Fig. 5 depicts results of proposed stacked autoencoders, with no overlap in MSE values between normal and abnormal samples, allowing for complete distinction with threshold.



(a) Kernel density of MSE (b) ROC curve

Fig. 5. The MSE and ROC of cascade stacked autoencoders

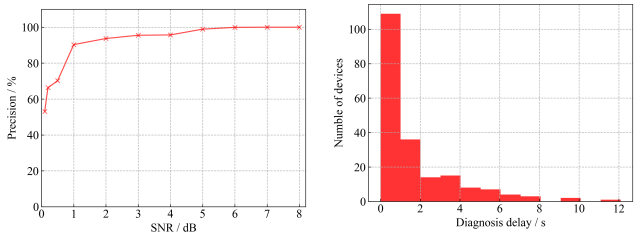
B. End-edge collaborative fault diagnosis

The previous experiments show 100% binary classification accuracy achieved for CWRU dataset using improved stacked

TABLE III
THE RESULT OF CASCADE STACKED AUTOENCODERS

Datasets	Healthy	Fault	FP	TN	Precision
Training set	2120	0	0	0	/
Test set	1592	7735	0	0	100.00%

autoencoder with only 24 input nodes. However, dataset uses high-accuracy sensors and circuits not practical for cost-sensitive industrial environments. Thus, added Gaussian noise to dataset and used for model training and inference to analyze impact of signal-to-noise ratio (SNR).



(a) Model accuracy at different SNR (b) Fault diagnosis delay

Fig. 6. Fault diagnosis based on end-edge collaboration.

Fig. 6(a) shows that the accuracy of inference decreases rapidly to an unusable level when SNR$<$1. Specifically, accuracy is 90.3% at SNR=1 and 100% at SNR$\geq$ 6. The proposed end-edge collaboration framework is validated using vibration signals with SNR=1 and end-model accuracy of 90.3%. Assuming 200 end devices under an edge node and the edge node can diagnosis 10 end nodes per second. It takes 20 s for the edge node to diagnose all end nodes once, with an average diagnosis latency of 10 s for each terminal node.

Based on Algorithm 1, 2, and 3, the kernel density plot of the diagnosis latency of edge node is showen in Fig. 6(b). Experimental results show that edge nodes efficiently prioritize high-risk end nodes with over 60% of samples' diagnosis latency less than 0.5 s. For all end nodes, the max diagnosis delay is less than 12 s, and the average latency is 0.597 s. Compare to sequential diagnosis tasks, the proposed end-edge collaborative algorithm can effectively reduce edge computing load by 94% while maintaining the same average diagnosis latency.

VI. CONCLUSION

Traditional rolling bearing fault diagnosis methods need to consume a large amount of computing resources. To address this problem, this paper proposes an end-edge collaborative bearing fault diagnosis framework, which can effectively utilize the remaining computing resource of the end device and greatly reduces the cost for using the fault diagnosis model. To address the problem of extremly limited resources and difficult model deployment, we propose stacked autoencoders, which adopt a reduce-breadth-with-depth approach to reduce memory consumption and achieve 100% accuracy of binary fault classification on the CWRU standard dataset by cascading two autoencoders, each with only 24 input nodes.

Considering the high sensor accuracy of CWRU dataset, the effects of different signal-to-noise ratios on model accuracy are compared by artificially adding Gaussian noise. The dynamic end-edge collaboration algorithms proposed are validated with a signal-to-noise ratio of 1 and the accuracy of end fault diagnosis 90.3%. The results show that the end-edge collaborative framework proposed effectively utilizes the residual computing resource of end devices in industrial systems and achieves high fault diagnosis accuracy at an exceptionally low cost.

Different from conventional model compression techniques (such as pruning and distillation), this paper effectively utilizes the difference in physical characteristics of normal and faulty bearing vibration signals, and proposes an ultra-lightweight model through an innovative network structure design. It means that in the future, we can still further compress the proposed stacked autoencoders by pruning and distillation methods.

ACKNOWLEDGMENT

This work was supported in part by the National Key R&D Program of China under Grant 2021YFB1715700, in part by the National Natural Science Foundation of China under Grant 62103046, 72192844 and 72201266, in part by the Fundamental Research Funds for the Central Universities under Grant E1E40805X2.

REFERENCES

[1] Y. He, M. Hu, K. Feng, and Z. Jiang, "Bearing Condition Evaluation Based on the Shock Pulse Method and Principal Resonance Analysis," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2021.

[2] R. N. Toma, A. E. Prosvirin, and J.-M. Kim, "Bearing Fault Diagnosis of Induction Motors Using a Genetic Algorithm and Machine Learning Classifiers," *Sensors*, vol. 20, no. 7, p. 1884, Jan. 2020.

[3] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, Mar. 2021.

[4] J. Lin, W.-M. Chen, Y. Lin, j. cohn, C. Gan, and S. Han, "MCUNet: Tiny Deep Learning on IoT Devices," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 11 711–11 722.

[5] H. Liu, W. Ziping, H. Zhang, B. Li, and C. Zhao, "Tiny Machine Learning (Tiny-ML) for Efficient Channel Estimation and Signal Detection," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2022.

[6] H. Ren, D. Anicic, and T. A. Runkler, "TinyOL: TinyML with Online-Learning on Microcontrollers," in *2021 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2021, pp. 1–8.

[7] C. Yang, Y. Wang, S. Lan, L. Wang, W. Shen, and G. Q. Huang, "Cloud-edge-device collaboration mechanisms of deep learning models for smart robots in mass personalization," *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102351, Oct. 2022.

[8] J. Shao, H. Zhang, Y. Mao, and J. Zhang, "Branchy-GNN: A Device-Edge Co-Inference Framework for Efficient Point Cloud Processing," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2021, pp. 8488–8492.

[9] W. Fang, F. Xue, Y. Ding, N. Xiong, and V. C. M. Leung, "EdgeKE: An On-Demand Deep Learning IoT System for Cognitive Big Data on Industrial Edge Devices," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 6144–6152, Sep. 2021.

[10] R. B. Randall and J. Antoni, "Rolling element bearing diagnostics—A tutorial," *Mechanical Systems and Signal Processing*, vol. 25, no. 2, pp. 485–520, Feb. 2011.

[11] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016.