IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 4, 2021, accepted April 6, 2021, date of publication April 12, 2021, date of current version April 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072596

Rolling Bearing Fault Diagnosis Method Based on Parallel QPSO-BPNN Under Spark-GPU Platform

LANJUN WAN[1,2], HONGYANG LI[1,2], GEN ZHANG[1,2], CHANGYUN LI[2], JUNFENG MAN[1,2], AND MANSHENG XIAO[1]
[1]School of Computer Science, Hunan University of Technology, Zhuzhou 412007, China
[2]Hunan Key Laboratory of Intelligent Information Perception and Processing Technology, Hunan University of Technology, Zhuzhou 412007, China

Corresponding author: Junfeng Man (manjunfeng@126.com)

This work was supported in part by the National Natural Science Foundation for Young Scientists of China under Grant 61702177, in part by the Natural Science Foundation of Hunan Province, China under Grant 2019JJ60048, Grant 2020JJ6086, and Grant 2019JJ60008; in part by the National Key Research and Development Project under Grant 2018YFB1700204 and Grant 2018YFB1003401, and in part by the Key Research and Development Project of Hunan Province under Grant 2019GK2133.

ABSTRACT Facing the massive rolling bearing vibration data, how to improve the training efficiency, diagnosis efficiency, and diagnosis accuracy of the rolling bearing fault diagnosis model is a challenge. Considering that the Spark-GPU platform provides powerful distributed parallel computing capabilities and back propagation neural network (BPNN) optimized by quantum particle swarm optimization (QPSO) algorithm has the characteristics of low computational complexity and high diagnosis accuracy, a rolling bearing fault diagnosis method based on parallel QPSO-BPNN under Spark-GPU platform is proposed. First, the distributed parallelization of QPSO-BPNN model based on Spark-GPU platform is realized, which can improve the training efficiency and diagnosis efficiency of rolling bearing fault diagnosis model in the big data environment. Second, in order to improve the convergence speed of fault diagnosis model, a parameter update strategy suitable for the distributed parallel training of QPSO-BPNN model is designed. At each iteration during training, the local parameters of each worker node are collected to the master node, and the global parameters are updated according to the weights and synchronized to each worker node. Third, a combination strategy of multiple QPSO-BPNN models based on ensemble learning is proposed. The weighted voting method is adopted to combine the output results of different QPSO-BPNN models to obtain the best fault diagnosis result of a sample, which can improve the fault diagnosis accuracy to a certain extent. Experimental results show that the proposed method can quickly perform model training and fault diagnosis for large-scale rolling bearing vibration data, and the fault diagnosis accuracy reaches 98.73%.

INDEX TERMS Back propagation neural network, fault diagnosis, GPU, parallelization, quantum particle swarm optimization, rolling bearing, Spark.

I. INTRODUCTION
Rolling bearing is one of the key components of mechanical equipment, and fault diagnosis of rolling bearing is essential to ensure long-term efficient and stable operation of mechanical equipment [1]. The traditional rolling bearing fault diagnosis methods based on signal processing technology have been widely used, such as enhanced singular spectrum decomposition [2], frequency phase space empirical wavelet transform [3], adaptive generalized demodulation [4], high-order synchrosqueezing transform [5], recycling variational mode decomposition [6], and resonance-based sparse signal decomposition [7], etc. The above methods can effectively diagnose rolling bearing faults when the time-frequency domain features of vibration signals are obvious. However, the signal processing technologies have certain limitations to deal with the complex vibration signals with noise and unobvious features.

In recent years, with the rapid development of machine learning and deep learning, there are more and more data-driven rolling bearing fault diagnosis methods based on machine learning and deep learning, such as naive bayes algorithm [8], least square support vector machine [9], iterative random forest [10], BP neural network [11], one-dimensional convolutional neural network [12], two-dimensional convolutional neural network [13], LSTM recurrent neural

The associate editor coordinating the review of this manuscript and approving it for publication was Asad Waqar Malik.

network [14], deep belief network [15], generative adversarial network [16], deep residual network [17], and transfer learning [18], etc. The above researches mainly focus on improving the accuracy, generalization, anti-noise ability, and adaptability of rolling bearing fault diagnosis model. They provide effective ways to mine the underlying fault features from the complex rolling bearing vibration signals, which can establish an effective mapping between the complex vibration signals and the output results of rolling bearing fault diagnosis model. However, the fault diagnosis models of rolling bearing based on machine learning and deep learning are generally complex and require a long time to be trained. Especially in the big data environment, the use of massive training samples requires a huge computational cost. The above studies seldom consider how to improve the training efficiency and diagnosis efficiency of rolling bearing fault diagnosis model in the big data environment.

With the rapid development of big data technology, many scholars have carried out extensive research on fault diagnosis in industrial big data scenarios [19]–[25]. For example, some studies combine big data technology and data-driven fault diagnosis methods to diagnose faults of mobile robot [21], sulfur hexafluoride electrical equipment [22], power grid equipment [23], wind turbine gearbox [24], and reciprocating air compressor [25]. Most of the above-mentioned studies use MapReduce [26] or Spark [27] to parallelize the fault diagnosis models to improve the performance of industrial equipment fault diagnosis in the big data environment. Compared with MapReduce, Spark introduces resilient distributed data set (RDD) and implements an efficient directed acyclic graph execution engine, it has a faster processing speed, and thus it is more suitable for efficient fault diagnosis in the big data environment.

Due to the many-core GPU has the advantages of high-performance, low-power, and low-cost, recently some work has been done to explore how to combine Spark and GPU to accelerate solving domain-specific applications, such as urban traffic vehicle recognition [28], magnetic resonance imaging [29], and remote sensing image processing [30], etc. The experimental results from [28]–[30] show that combining Spark and GPU can significantly improve the performance of these applications, but the implementations of them are complicated and it is difficult to port their implementation methods to other fields. The newly released Spark 3.0 already supports the accelerator-aware scheduling, allowing users to discover and request GPU computing resources at Executor, Driver, and Task levels, which simplifies the development of applications based on Spark and GPU.

The authors' previous work [31] proposed a rolling bearing fault diagnosis method based on QPSO-BPNN and Dempster-Shafer evidence theory, which can effectively and accurately diagnose different types of rolling bearing faults under different working conditions. With the expansion of industrial production scale and the increase in the complexity of mechanical equipment, the vibration data of rolling bearing collected by multiple sensors in real time are growing rapidly in the actual production environment. It is difficult to efficiently perform model training and fault diagnosis for large-scale rolling bearing vibration data using the serial QPSO-BPNN proposed in the previous work. For the massive rolling bearing vibration data, how to improve the training efficiency and diagnosis efficiency of rolling bearing fault diagnosis model is an urgent problem to be solved.

The authors' another previous work [32] proposed a rolling bearing fault diagnosis method based on Spark and ACO-K-Means clustering algorithm, the ACO-K-Means clustering algorithm is successfully parallelized on Spark platforms, which can efficiently carry out clustering analysis on the massive rolling bearing vibration data in parallel. The proposed method focuses on improving the model training efficiency and fault diagnosis efficiency by fully utilizing all available CPU and memory resources on a Spark cluster. Compared with Spark platform, Spark-GPU platform has stronger distributed parallel computing ability, thus it is more helpful for improving the model training efficiency and fault diagnosis efficiency. Compared with ACO-K-Means clustering algorithm, QPSO-BPNN with strong non-linear mapping ability, high self-learning and adaptive abilities can obtain a higher and more stable fault diagnosis accuracy.

Therefore, a rolling bearing fault diagnosis method based on parallel QPSO-BPNN under Spark-GPU platform is proposed, which aims to fully exploit the powerful distributed parallel computing capabilities provided by the Spark-GPU platform and take advantage of QPSO-BPNN with low computational complexity and high diagnosis accuracy to achieve more efficient and accurate fault diagnosis of rolling bearing in the big data environment. However, it is still a challenge to efficiently implement QPSO-BPNN on a Spark-GPU platform. The current work focuses on how to efficiently perform model training and fault diagnosis for large-scale rolling bearing vibration data using the parallel QPSO-BPNN on Spark-GPU platforms.

The main contributions of this paper are summarized as follows.

- The distributed parallelization of QPSO-BPNN model based on Spark-GPU platform is realized, which significantly improves the training efficiency and diagnosis efficiency of rolling bearing fault diagnosis model based on QPSO-BPNN in the big data environment.
- A parameter update strategy suitable for the distributed parallel training of QPSO-BPNN model is proposed. At each iteration during training, the local parameters of each worker node are collected to the master node, and the global parameters are updated according to the weights and synchronized to each worker node, which improves the convergence speed of rolling bearing fault diagnosis model in the distributed parallel environment.
- A combination strategy of multiple QPSO-BPNN models based on ensemble learning is proposed. The output results of QPSO-BPNN models respectively corresponding to the base end, drive end, and fan end of rolling bearing are combined by weighted voting to

obtain the best fault diagnosis result of a sample, which improves the fault diagnosis accuracy of rolling bearing to a certain extent.

- The effectiveness of the proposed rolling bearing fault diagnosis method is verified by a large number of experiments. Experimental results show that this method can not only make full use of the computing resources of the Spark-GPU platform to quickly perform model training and fault diagnosis on the massive rolling bearing vibration data, but also obtain a higher fault diagnosis accuracy.

The rest of this paper is organized as follows. The QPSO-BPNN model is introduced in Section II. The rolling bearing fault diagnosis method based on parallel QPSO-BPNN under Spark-GPU platform is described in Section III. The experimental results and analysis are presented in Section IV. The conclusions and future work are given in Section V.

II. THE PREVIOUS QPSO-BPNN MODEL

BPNN [33] is a classic multi-layer feedback forward neural network, which is characterized by forward propagation of signals and backward propagation of errors. Because it has a simple network structure and a strong nonlinear mapping ability, it is widely used to handle classification problems. The training of BPNN is divided into two stages: signal propagation, and weights and thresholds update. In the first stage, at first the signals are propagated from the input layer to the hidden layer, then the signals are propagated to the output layer in turn according to the weights and activation function of each neuron in the hidden layer, and finally the output results are obtained. In the second stage, at first the errors between the output results and targets are calculated and backward propagated, and then the weights and threshold of each neuron in each layer are corrected according to the errors. The above two stages are executed iteratively to complete the training of BPNN. However, BPNN has the disadvantages of slow convergence speed and easy to trap in local minimums, this is mainly because BPNN randomly initializes the weights and threshold of each neuron. As a result, recently some researchers began to adopt intelligent optimization algorithms to improve the initialization of the weights and thresholds of BPNN, such as genetic algorithm (GA) [34], differential evolution (DE) algorithm [35], and particle swarm optimization (PSO) algorithm [36], etc.

The classic PSO algorithm [37] is a swarm intelligence random search algorithm, which searches for the optimal solution according to the optimal particle in the solution space through iteration. However, since PSO algorithm has the problem of easily falling into a local optimal solution, recently some improved PSO algorithms have been developed, such as adaptive particle swarm optimization (APSO) algorithm [38], selective particle swarm optimization (SPSO) algorithm [39], and QPSO algorithm [40], etc. APSO algorithm adopts the nonlinear function to dynamically adjust the inertia weights and the contribution of each particle to avoid falling into the local optimal solution. SPSO algorithm changes the search space from a real-valued space into a set of selected values, which can reduce the computational cost of fitness values. QPSO algorithm mainly improves the PSO algorithm as follows. For one thing, the position of each particle is updated according to the average best position of quantum particle swarm, and the moving speed of the particle is no longer considered, which increases the randomness of the particle movement, so it can avoid falling into the local optimal solution. For another thing, only the shrinkage factor that controls the update of the particle position needs to be tuned, which is easier for performance tuning and enhances the global convergence ability. Compared with APSO algorithm and SPSO algorithm, QPSO algorithm can obtain a faster convergence speed with less computational cost, and it is more likely to obtain the optimal initial weights and thresholds of BPNN due to it has a stronger global convergence ability. Therefore, QPSO algorithm is more suitable for optimizing the initial weights and thresholds of BPNN.

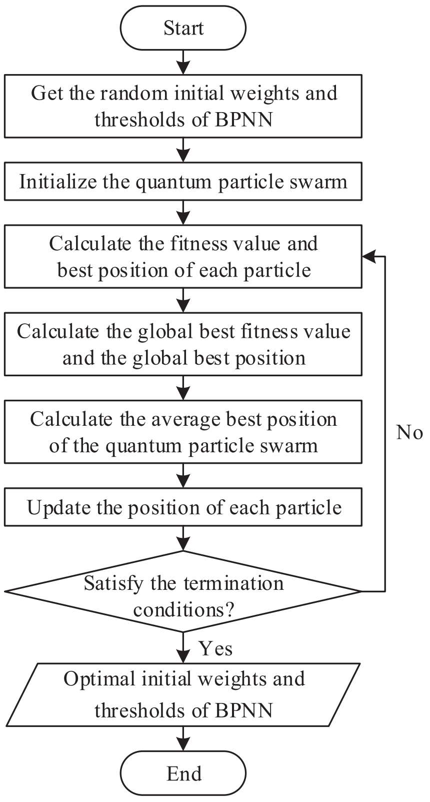
Start
Get the random initial weights and thresholds of BPNN
Initialize the quantum particle swarm
Calculate the fitness value and best position of each particle
Calculate the global best fitness value and the global best position
Calculate the average best position of the quantum particle swarm
Update the position of each particle
Satisfy the termination conditions?
No
Yes
Optimal initial weights and thresholds of BPNN
End

FIGURE 1. Process of optimizing the initial weights and thresholds of BPNN with QPSO algorithm.

The authors' previous work [31] adopted QPSO algorithm to optimize the initial weights and thresholds of BPNN, as shown in Fig. 1. First, the random initial weights and thresholds of BPNN are obtained and the quantum particle swarm is initialized, including the number of particles, the dimension of particles, and the initial position of each particle (i.e., the initial weights and thresholds of BPNN). Second, the position of each particle is used as the weights

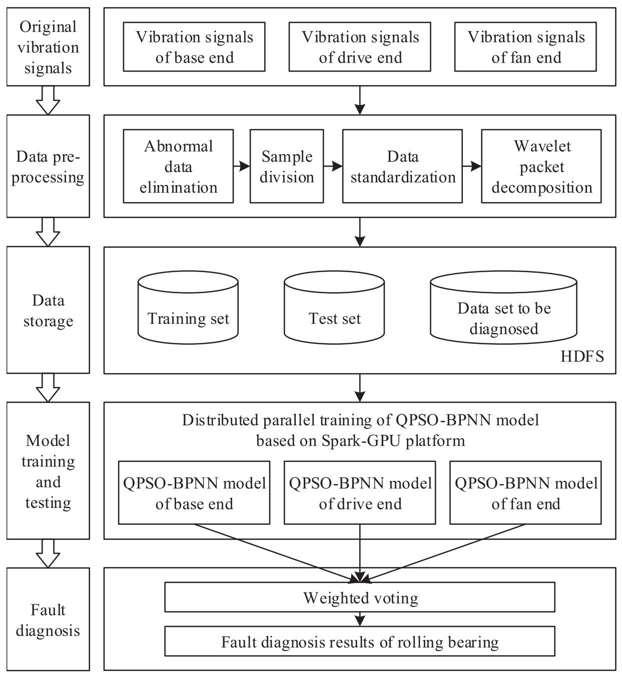
Original vibration signals
Vibration signals of base end
Vibration signals of drive end
Vibration signals of fan end
Data pre-processing
Abnormal data elimination
Sample division
Data standardization
Wavelet packet decomposition
Data storage
Training set
Test set
Data set to be diagnosed
HDFS
Model training and testing
Distributed parallel training of QPSO-BPNN model based on Spark-GPU platform
QPSO-BPNN model of base end
QPSO-BPNN model of drive end
QPSO-BPNN model of fan end
Fault diagnosis
Weighted voting
Fault diagnosis results of rolling bearing

FIGURE 2. Process of rolling bearing fault diagnosis based on parallel QPSO-BPNN under Spark-GPU platform.

and thresholds of BPNN to train the BPNN model once. Third, the fitness value and best position of each particle are calculated, where the fitness value of each particle is the error of the BPNN model training. Fourth, the global best fitness value and the global best position of quantum particle swarm are calculated. Fifth, the average best position of quantum particle swarm is calculated. Sixth, the position of each particle is updated according to the best position of each particle, the global best position and average best position of quantum particle swarm, and the shrinkage factor. Finally, determining whether the max iterations is reached or a satisfactory solution is obtained, if so, the optimal initial weights and thresholds of BPNN are obtained; otherwise, the next iteration is continue to be executed.

III. THE PROPOSED FAULT DIAGNOSIS METHOD OF ROLLING BEARING

A. PROCESS OF ROLLING BEARING FAULT DIAGNOSIS BASED ON PARALLEL QPSO-BPNN

The overall process of rolling bearing fault diagnosis based on parallel QPSO-BPNN under Spark-GPU platform is shown in Fig. 2, which includes the following four stages: data preprocessing, data storage, model training and testing, and fault diagnosis.

In the data preprocessing stage, first, the abnormal data contained in the original vibration signals collected by the sensors deployed on the base end, drive end, and fan end of rolling bearing are eliminated. Second, the cleaned data are divided into several samples. Third, each sample is standardized. Finally, the wavelet packet decomposition is performed on each sample to obtain the eigenvectors of different running states of rolling bearing.

In the data storage stage, all eigenvectors are stored in Hadoop Distributed File System (HDFS), and the eigenvectors used for model training and testing are divided into training set and test set.

In the model training and testing stage, first, the network structures and training parameters of QPSO-BPNN models respectively corresponding to the base end, drive end, and fan end are determined. Second, the training samples corresponding to the base end, drive end, and fan end from HDFS are used as the input of these three models respectively, and the distributed parallel trainings of these three models are performed on Spark-GPU platform. Finally, the test set from HDFS is used to test the three models.

In the fault diagnosis stage, first, the data to be diagnosed are read from HDFS. Second, the above three trained QPSO-BPNN models are performed to diagnose these data on Spark-GPU platform, respectively. Finally, the weighted voting method is adopted to combine the output results of these three models to obtain the final fault diagnosis results.

B. PARALLEL DESIGN OF QPSO-BPNN MODEL BASED ON SPARK-GPU PLATFORM

1) OVERALL DISTRIBUTED PARALLEL DESIGN SCHEME

According to the idea of data parallelism, the overall distributed parallel design scheme of QPSO-BPNN model based on Spark-GPU platform is proposed, as shown in Fig. 3.

On a Spark-GPU platform, the master node is responsible for the task scheduling and resource management of the entire cluster, and each worker node can use one or more Spark executors to train one or more QPSO-BPNN models. Each Spark executor can exploit the RAPIDS library [41] developed by means of CUDA to call GPU computing resources to accelerate the training of QPSO-BPNN model. When starting the QPSO-BPNN model training program on Spark-GPU platform, firstly, a SparkContext object is initialized on the master node; secondly, the rolling bearing training set is read from HDFS to create an RDD; thirdly, each Spark executor reads the data of an RDD partition to train a QPSO-BPNN model. The distributed parallel training of QPSO-BPNN model based on Spark-GPU platform mainly includes the following two stages.

In the first stage, the QPSO algorithm is executed in parallel to optimize the initial weights and thresholds of BPNN. In each iteration of QPSO algorithm, firstly, the fitness value and best position of each particle are calculated. Due to the computational tasks of different particles are independent of each other, the computational tasks of all particles can be reasonably allocated to each Spark executor, and multiple Spark executors can be used to perform computational tasks of different particles in parallel. Secondly, the fitness value and best position of each particle are collected to calculate the global best fitness value, global best position, and average best position of quantum particle swarm. Finally, the position of each particle is updated according to the best position of each particle, the average best position of quantum particle swarm, and the shrinkage factor. Due to the updating

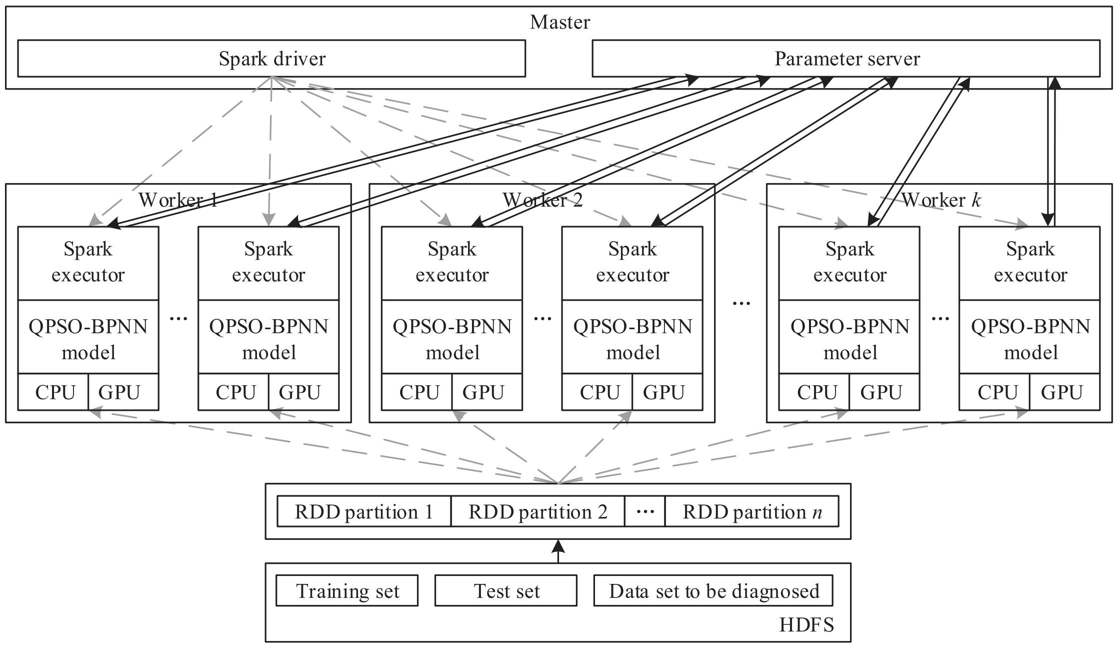

FIGURE 3. Overall distributed parallel design scheme of QPSO-BPNN model based on Spark-GPU platform.

processes of different particles are independent of each other, the position of each particle can be updated in parallel.

In the second stage, the QPSO-BPNN model is trained in parallel. After obtaining the optimal initial weights and thresholds of BPNN, the data-parallel strategy is adopted to realize the parallel training of QPSO-BPNN model. A large-scale training set is divided into n smaller training subsets, and one training subset is used to train one QPSO-BPNN model. Due to the training of each model is independent of each other, multiple Spark executors can be used to perform model training tasks in parallel. In each iterative training of QPSO-BPNN model, firstly, each Spark executor replaces the weights and thresholds of the current model with the new global weights and thresholds. Secondly, each Spark executor uses its own training subset to perform the model training once to obtain the new weights and thresholds. Finally, the weights and thresholds of all models are collected to update the global weights and thresholds.

Both the calculation of the fitness value of each particle in the first stage and the training of each QPSO-BPNN model in the second stage involve a large number of matrix operations, thus it is very suitable to use GPU to speed up the training of the entire model.

2) DESIGN OF PARAMETER UPDATE STRATEGY

In the distributed parallel training of QPSO-BPNN model, using the idea of the parameter server architecture [42], the master node is used as the parameter server node to collect the weights and thresholds of QPSO-BPNN model of each worker node on Spark-GPU platform, the global weights and thresholds are updated according to the weight of each model, and the updated global weights and thresholds are synchronized to each worker node.

In the t-th iterative training of QPSO-BPNN model, the process of parameter update includes the following steps.

Step 1. On the k worker nodes, the n QPSO-BPNN models corresponding to n training subsets are trained in parallel according to the current weights and thresholds $(g_1^t, g_2^t, \ldots, g_n^t)$, and the new weights and thresholds $(g_{1.temp}^t, g_{2.temp}^t, \ldots, g_{n.temp}^t)$ and the losses $(loss_1^t, loss_2^t, \ldots, loss_n^t)$ of n QPSO-BPNN models are obtained and collected to the parameter server node, where g_i^t represents the weights and thresholds used for training the i-th QPSO-BPNN model and $1 \le i \le n$.

Step 2. On the parameter server node, firstly, the losses of all QPSO-BPNN models are normalized by Min-Max normalization method. Secondly, according to the normalized losses $(l_1^t, l_2^t, \ldots, l_n^t)$, the weight of each model in the global parameter update is calculated by

$$\eta_i = (1 - l_i^t) / \sum_{j=1}^{n} 1 - l_j^t, \tag{1}$$

where η_i represents the weight of the i-th QPSO-BPNN model and l_i^t represents the normalized result of the loss obtained after the t-th iterative training of the i-th QPSO-BPNN model. Thirdly, according to the weight η_i, the global weights and thresholds are updated by

$$G^{t+1} = G^t + \sum_{i=1}^{n} \eta_i \left(g_{i.temp}^t - G^t \right), \tag{2}$$

where G^t denotes the current global weights and thresholds.

Step 3. The parameter server node broadcasts the new global weights and thresholds G^{t+1} to all worker nodes, and the weights and thresholds of all QPSO-BPNN models on

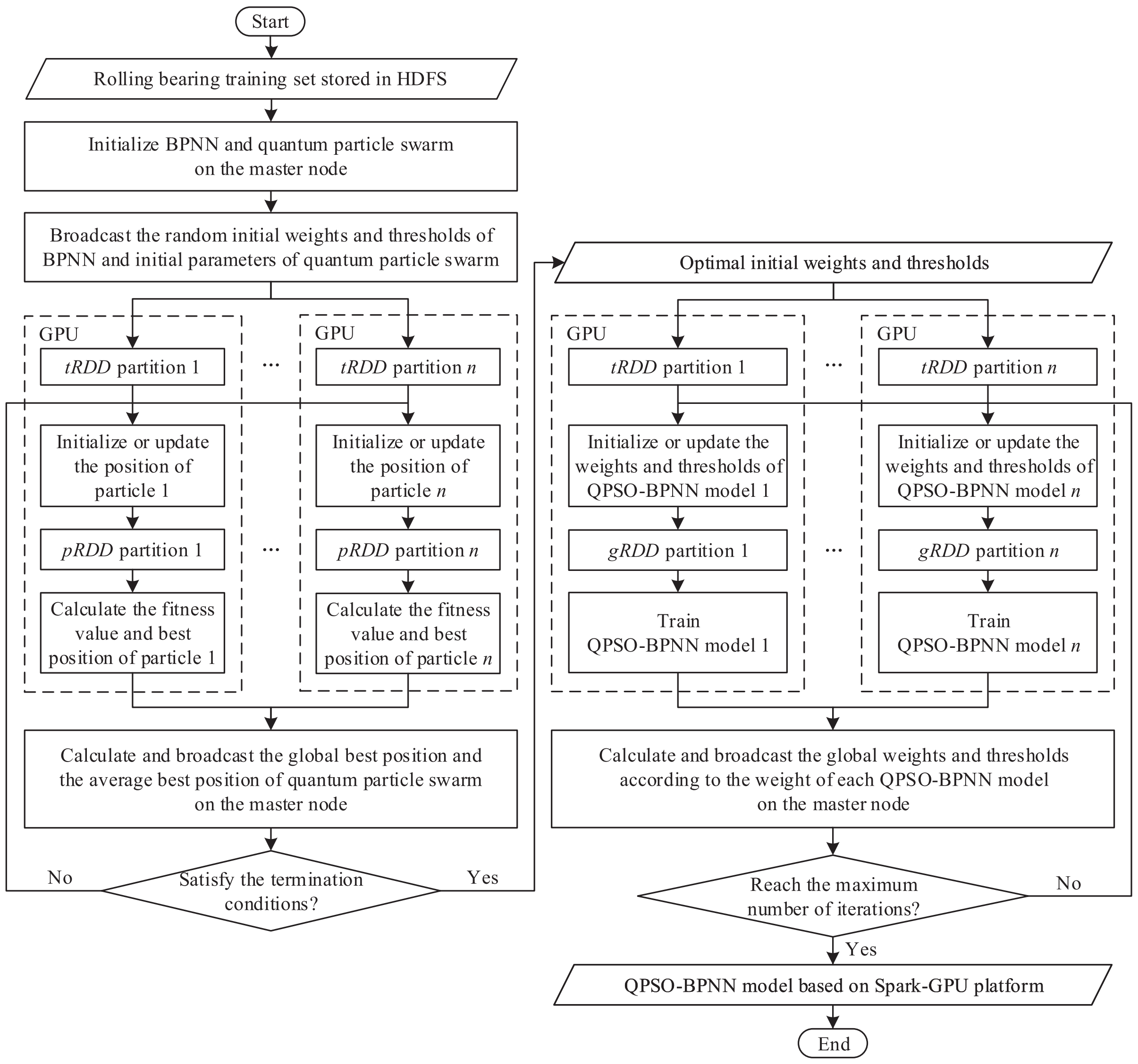
Start
Rolling bearing training set stored in HDFS
Initialize BPNN and quantum particle swarm on the master node
Broadcast the random initial weights and thresholds of BPNN and initial parameters of quantum particle swarm
GPU
tRDD partition 1
tRDD partition n
Initialize or update the position of particle 1
Initialize or update the position of particle n
pRDD partition 1
pRDD partition n
Calculate the fitness value and best position of particle 1
Calculate the fitness value and best position of particle n
Calculate and broadcast the global best position and the average best position of quantum particle swarm on the master node
Satisfy the termination conditions?
No
Yes
Optimal initial weights and thresholds
Initialize or update the weights and thresholds of QPSO-BPNN model 1
Initialize or update the weights and thresholds of QPSO-BPNN model n
gRDD partition 1
gRDD partition n
Train QPSO-BPNN model 1
Train QPSO-BPNN model n
Calculate and broadcast the global weights and thresholds according to the weight of each QPSO-BPNN model on the master node
Reach the maximum number of iterations?
QPSO-BPNN model based on Spark-GPU platform
End

FIGURE 4. Flowchart of parallel implementation of QPSO-BPNN model based on Spark-GPU platform.

each worker node are updated synchronously, i.e., $g_1^{t+1} = g_2^{t+1} = \cdots = g_n^{t+1} = G^{t+1}$.

C. PARALLEL IMPLEMENTATION OF QPSO-BPNN MODEL BASED ON SPARK-GPU PLATFORM

According to the above-mentioned distributed parallel design scheme of QPSO-BPNN model based on Spark-GPU platform, this subsection describes the parallel implementation of QPSO-BPNN model based on Spark-GPU platform. The flowchart of parallel implementation of QPSO-BPNN model based on Spark-GPU platform is shown in Fig. 4 and the pseudo-code of that is described in Algorithm 1, which mainly includes the following two stages.

The first stage is the parallel implementation of QPSO algorithm for optimizing the initial weights and thresholds of BPNN based on Spark-GPU platform, including the following steps.

Step 1. Initialize BPNN and quantum particle swarm on the master node. Firstly, the network structure of BPNN needs to be determined. The number of the input layer nodes is set to 8 which is the dimension of an eigenvector. The number of the hidden layers is set to 2, and the number of the first and second hidden layer nodes are set to 20 and 12, respectively, which are determined by Hofferding inequality and [43]. The number of the output layer nodes is set to 4 which is the number of classes of rolling bearing running states. Secondly, the weights and thresholds G^1 of BPNN are randomly initialized, and the particle number n of quantum particle swarm and the dimension of particle are initialized. The dimension of particle is determined by the number of weights and thresholds of the input layer, hidden layer, and output layer of BPNN, i.e., $(8 \times 20 + 20) + (20 \times 12 + 12) + (12 \times 4 + 4) = 484$. Finally, the random initial weights and thresholds of BPNN and initial parameters of quantum particle swarm are broadcasted to all worker nodes.

Algorithm 1 Parallel Implementation of QPSO-BPNN Model Based on Spark-GPU Platform

```
Require: $m$ eigenvectors, the number of particles $n$, the max iterations $maxIterQPSO$ and error goal $errGoal$ of QPSO, the max iterations $maxIterBPNN$ of BPNN
Ensure: The trained QPSO-BPNN model
1: Initialize the weights and thresholds $G^1$ of BPNN and quantum particle swarm on the master node;
2: Broadcast $G^1$ and the initial parameters of quantum particle swarm to each worker node;
3: Read the training set with $m$ eigenvectors from HDFS to create an RDD in parallel: $tRDD = (E_1, E_2, \ldots, E_m)$;
4: for $i \leftarrow 1$ to $maxIterQPSO$ do
5:     for all Spark executors on GPUs in parallel do
6:         if $i = 1$ then
7:             Initialize the positions: $(P_1^1, P_2^1, \ldots, P_n^1) \leftarrow G^1$;
8:         else
9:             Update the positions $(P_1^i, P_2^i, \ldots, P_n^i)$ by (3);
10:        end if
11:        Create or update the key-value pair RDD: $pRDD = (\langle P_1^i, E_1\rangle, \langle P_1^i, E_2\rangle, \ldots, \langle P_n^i, E_m\rangle)$;
12:        Calculate the fitness values of all particles: $(f_1^i, f_2^i, \ldots, f_n^i) \leftarrow \mathrm{BPNN}((P_1^i, P_2^i, \ldots, P_n^i), pRDD)$;
13:        Calculate the best positions of all particles: $(P_{1.best}^i, P_{2.best}^i, \ldots, P_{n,best}^i) \leftarrow \mathrm{cmp}((f_1^i, f_2^i, \ldots, f_n^i), (f_1^{i-1}, f_2^{i-1}, \ldots, f_n^{i-1}))$;
14:    end for
15:    Calculate the global best fitness value on the master node: $f_{best}^i \leftarrow \min(f_1^i, f_2^i, \ldots, f_n^i)$;
16:    Calculate the global best position on the master node: $P_{best}^i \leftarrow P^i_{(\arg\min_{x\in\{1,2,\ldots,n\}} f_x^i).best}$;
17:    if $i \geq 2$ and $f_{best}^{i-1} \leq f_{best}^i$ then
18:        $P_{best}^i \leftarrow P_{best}^{i-1}, f_{best}^i \leftarrow f_{best}^{i-1}$;
19:    end if
20:    Calculate the average best position on the master node: $M_{best}^i \leftarrow \sum_{x=1}^n P_{x.best}^i / n$;
21:    Broadcast $P_{best}^i$ and $M_{best}^i$ to all worker nodes;
22:    if $f_{best}^i < errGoal$ then break; end if
23: end for
24: Get the best initial weights and thresholds: $G^1 \leftarrow P_{best}^i$;
25: for $i \leftarrow 1$ to $maxIterBPNN$ do
26:    for all Spark executors on GPUs in parallel do
27:        Initialize or update the weights and thresholds of $n$ QPSO-BPNN models: $(g_1^i, g_2^i, \ldots, g_n^i) \leftarrow G^i$;
28:        Create or update the key-value pair RDD: $gRDD = (\langle g_1^i, E_1\rangle, \langle g_1^i, E_2\rangle, \ldots, \langle g_n^i, E_m\rangle)$;
29:        Train $n$ QPSO-BPNN models: $(g_{1.temp}^i, g_{2.temp}^i, \ldots, g_{n.temp}^i) \leftarrow \mathrm{QPSO\_BPNN}((g_1^i, g_2^i, \ldots, g_n^i), gRDD)$;
30:    end for
31:    Update the global weights and thresholds $G^{i+1}$ by (1) and (2) on the master node;
32:    Broadcast $G^{i+1}$ to all worker nodes;
33: end for
```

Step 2. Read the training set from HDFS and use multiple Spark executors to call GPU computing resources to create an RDD *tRDD* in parallel, where each element of *tRDD* is an eigenvector. *tRDD* can be equally divided into n RDD partitions according to the particle number n, if *tRDD* contains m eigenvectors, then the x-th partition of *tRDD* can be denoted by $(E_{(x-1)m/n+1}, E_{(x-1)m/n+2}, \ldots, E_{xm/n})$, where $1 \leq x \leq n$.

Step 3. Use multiple Spark executors to call GPU computing resources to initialize the positions of all particles and create a new RDD in parallel. Firstly, G^1 is adopted to initialize the positions $(P_1^1, P_2^1, \ldots, P_n^1)$ of all particles, i.e., $P_1^1 = P_2^1 = \cdots = P_n^1 = G^1$. Secondly, a key-value pair RDD *pRDD* is constructed by taking the position of each particle as a key and each eigenvector of *tRDD* as a value, and the x-th partition of *pRDD* can be represented by $(\langle P_x^1, E_{(x-1)m/n+1}\rangle, \langle P_x^1, E_{(x-1)m/n+2}\rangle, \ldots, \langle P_x^1, E_{xm/n}\rangle)$, where $1 \leq x \leq n$.

Step 4. Use multiple Spark executors to call GPU computing resources to calculate the fitness values and best positions of all particles in parallel. Firstly, the data of each partition in *pRDD* is used to train a BPNN model respectively, and each model is trained once to obtain the fitness values $(f_1^i, f_2^i, \ldots, f_n^i)$, where f_x^i denotes the fitness value obtained by the x-th particle in the i-th iteration. Secondly, the best positions $(P_{1.best}^i, P_{2.best}^i, \ldots, P_{n.best}^i)$ of all particles are determined by comparing the fitness values $(f_1^i, f_2^i, \ldots, f_n^i)$ obtained by all particles in the current iteration with the fitness values $(f_1^{i-1}, f_2^{i-1}, \ldots, f_n^{i-1})$ obtained by all particles in the previous iteration. If $i \geq 2$ and $f_x^{i-1} \leq f_x^i$, then $P_{x.best}^i = P_x^{i-1}$; otherwise, $P_{x.best}^i = P_x^i$, where $1 \leq x \leq n$.

Step 5. Calculate and broadcast the global best position and average best position of quantum particle swarm on the master node. Firstly, the master node collects the fitness values and best positions of all particles. Secondly, the global best fitness value $f_{best}^i = \min(f_1^i, f_2^i, \ldots, f_n^i)$ and the global best position $P_{best}^i = P^i_{(\arg\min_{x\in\{1,2,\ldots,n\}} f_x^i).best}$ of quantum particle swarm are obtained by comparing the fitness values of all particles in the i-th iteration. Thirdly, the global best position of quantum particle swarm is updated by comparing the global best fitness value f_{best}^i obtained in the current iteration with the global best fitness value f_{best}^{i-1} obtained in the previous iteration. If $i \geq 2$ and $f_{best}^{i-1} \leq f_{best}^i$, then $P_{best}^i = P_{best}^{i-1}$ and $f_{best}^i = f_{best}^{i-1}$; otherwise, there is no need to update the global best position. Fourthly, the average best position of quantum particle swarm is calculated as $M_{best}^i = \sum_{x=1}^{n} P_{x.best}^i / n$. Finally, P_{best}^i and M_{best}^i are broadcasted to all worker nodes.

Step 6. Determine whether the current iteration number reaches the max iterations or whether the global best fitness value is lower than the error goal. If so, the iteration is terminated and the optimal initial weights and thresholds of BPNN are returned, i.e., $G^1 = P_{best}^i$; otherwise, the positions of all particles are updated and the next iteration will be continued by going back to Step 4. The process of using multiple Spark executors to call GPU computing resources to update the positions of all particles and *pRDD* in parallel is as follows. Firstly, according to the best position of all particles and the global best position and average best position of quantum particle swarm, the latest positions $(P_1^{i+1}, P_2^{i+1}, \ldots, P_n^{i+1})$ of all particles are calculated by

$$P_x^{i+1} = \alpha P_{x.best}^i + (1-\alpha)P_{best}^i \pm \varphi \left| M_{best}^i - P_x^i \right| \ln \frac{1}{\beta}, \quad (3)$$

where P_x^{i+1} represents the latest position of the x-th particle in the $(i+1)$-th iteration, α and β are uniform distributions on (0, 1), and φ represents the shrinkage factor [44]. To increase the randomness of the particle movement, $\pm$ is used before the absolute term that is the distance between the average best position of quantum particle swarm and the latest position of the particle. Secondly, Min-Max normalization is performed on the latest positions of all particles, and the key of each key-value pair in *pRDD* is updated accordingly.

The second stage is the parallel implementation of QPSO-BPNN model training based on Spark-GPU platform, including the following steps.

Step 1. Use multiple Spark executors to call GPU computing resources to initialize the weights and thresholds of all QPSO-BPNN models and create a new RDD in parallel. Firstly, the optimal initial weights and thresholds G^1 are used as the initial weights and thresholds $(g_1^1, g_2^1, \ldots, g_n^1)$ of all QPSO-BPNN models, i.e., $g_1^1 = g_2^1 = \cdots = g_n^1 = G^1$. Secondly, a key-value pair RDD *gRDD* is constructed by taking the weights and thresholds of each QPSO-BPNN model as a key and each eigenvector of *tRDD* as a value, and the x-th partition of *gRDD* can be denoted by $(\langle g_x^1, E_{(x-1)m/n+1}\rangle, \langle g_x^1, E_{(x-1)m/n+2}\rangle, \ldots, \langle g_x^1, E_{xm/n}\rangle)$, where $1 \le x \le n$.

Step 2. Use multiple Spark executors to call GPU computing resources to train all QPSO-BPNN models in parallel. The data of each partition in *gRDD* is used to train a QPSO-BPNN model respectively, and each model is trained once to obtain the latest weights and thresholds $(g_{1.temp}^i, g_{2.temp}^i, \ldots, g_{n.temp}^i)$, where $g_{x.temp}^i$ denotes the latest weights and thresholds of the x-th QPSO-BPNN model obtained in the i-th iteration.

Step 3. Calculate and broadcast the global weights and thresholds according to the weight of each QPSO-BPNN model on the master node. Firstly, the master node collects the latest weights and thresholds of all QPSO-BPNN models. Secondly, the global weights and thresholds G^{i+1} are updated by (1) and (2) and broadcasted to all worker nodes.

Step 4. Determine whether the current iteration number reaches the max iterations. If so, the iteration is terminated and the final QPSO-BPNN model is obtained; otherwise, the weights and thresholds of all QPSO-BPNN models are updated and the next iteration will be continued by going back to Step 2. The process of using multiple Spark executors to call GPU computing resources to update the weights and thresholds and *gRDD* in parallel is as follows. Firstly, the weights and thresholds $(g_1^{i+1}, g_2^{i+1}, \ldots, g_n^{i+1})$ of all QPSO-BPNN models are updated according to G^{i+1}, i.e., $g_1^{i+1} = g_2^{i+1} = \cdots = g_n^{i+1} = G^{i+1}$. Secondly, the key of each key-value pair in *gRDD* is updated as the new weights and thresholds.

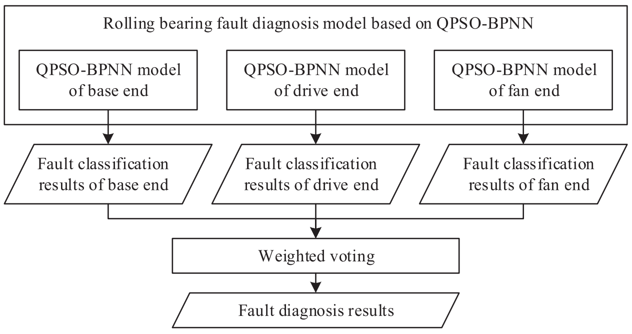
Rolling bearing fault diagnosis model based on QPSO-BPNN
QPSO-BPNN model of base end
QPSO-BPNN model of drive end
QPSO-BPNN model of fan end
Fault classification results of base end
Fault classification results of drive end
Fault classification results of fan end
Weighted voting
Fault diagnosis results

FIGURE 5. Combination of three QPSO-BPNN models based on ensemble learning.

D. COMBINATION STRATEGY OF MULTIPLE QPSO-BPNN MODELS BASED ON ENSEMBLE LEARNING

The proposed rolling bearing fault diagnosis model is composed of QPSO-BPNN models respectively corresponding to the base end, drive end, and fan end, which can reduce the risk of misdiagnosis caused by the wrong classification of a single QPSO-BPNN model, and can improve the fault diagnosis accuracy to a certain extent. The common combination strategies include Dempster-Shafer (DS) evidence theory [45] and ensemble learning [46]. Considering that ensemble learning can avoid the explosive growth of the exponential function and the problem of more parameters required for calculating the basic probability distribution function in DS evidence theory, a combination strategy of multiple QPSO-BPNN models based on ensemble learning is proposed, as shown in Fig. 5.

In the combination of multiple QPSO-BPNN models based on ensemble learning, the weighted voting method is used to combine the classification results of multiple basic classifiers (i.e., QPSO-BPNN models) to obtain the best fault diagnosis result of a sample. The classification results of multiple basic classifiers for sample x are combined by weighted voting according to

$$H(x) = \underset{y\in\{1,2,\ldots,j\}}{\arg\max} \sum_{i=1}^{s} \omega_i^y h_i^y(x), \quad (4)$$

and

$$\omega_i^y = Acc_i^y / \sum_{k=1}^{s} Acc_k^y, \quad (5)$$

where j represents the number of classes, s represents the number of basic classifiers, ω_i^y represents the weight of the i-th basic classifier when classifying the sample x into class y, $h_i^y(x)$ denotes the probability that the i-th basic classifier classifies the sample x into class y, and Acc_i^y denotes the accuracy of the i-th basic classifier to classify the sample whose true classification result is class y into class y. During the fault diagnosis, the running states of rolling bearing include

TABLE 1. Hardware environment of the cluster.

Node Type	Number of Nodes	CPU Model	GPU Model	CPU Cores Per Node	GPU Cores Per Node	Host Memory Per Node (GB)	GPU Memory Per Node (GB)
Master node	1	Intel Xeon E3-1225 v5 @3.3GHz	–	4	–	32	–
Worker node	4	Intel Core i7-9700K @3.6GHz	NVIDIA GeForce RTX 2070 SUPER	8	2560	32	8

TABLE 2. Software environment of the cluster.

Software Name	Software Version
Python	Python 3.8.2
Scala	Scala 2.12.10
CUDA toolkit	CUDA 10.2.89
Spark	Spark 3.0.0
RAPIDS	Rapids 4 Spark
Hadoop	Hadoop 3.2.1
Linux operating system	CentOS 8.1

TABLE 3. Description of the rolling bearing data set.

Data Set Name	Data Set Size (GB)	Ratio of Trainin Set to Test Set	Number of Samples in Training Set	Number of Samples in Test Set
DataSet 1	8	8:2	7.09×10^7	1.77×10^7
DataSet 2	16	8:2	1.42×10^8	3.54×10^7
DataSet 3	32	8:2	2.83×10^8	7.09×10^7

normal state, inner race fault, ball fault, and outer race fault, thus j can be set to 4; three basic classifiers (i.e., QPSO-BPNN models respectively corresponding to the base end, drive end, and fan end) are used, thus s can be set to 3.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. EXPERIMENTAL SETUP

The experimental platform used in this paper is a distributed cluster. The hardware environment of the cluster is shown in Table. 1, and the software environment of the cluster is shown in Table. 2. In order to compare and analyze the impact of using GPU and not using GPU on the fault diagnosis accuracy, model training efficiency, and fault diagnosis efficiency, a series of experiments are carried out on the experimental platform with GPU (called Spark-GPU platform) and the experimental platform without GPU (called Spark platform).

The experimental data used in this paper are the vibration data of rolling bearing in different running states provided by the Case Western Reserve University Bearing Data Center [47]. They are collected by sensors deployed on the base end, drive end, and fan end of rolling bearing under different working conditions. Due to a large-scale data set is more helpful to verify the effectiveness of the proposed fault diagnosis method, at first the sliding window method [48] is adopted to enhance the original vibration data, then the enhanced data are preprocessed (see Section III-A), and finally the three different size data sets composed of eigenvectors are obtained. Table. 3 presents the description of the rolling bearing data set. Each data set includes the following different running-state monitoring data of rolling bearing: normal state data, inner race fault data, ball fault data, and outer race fault data. Each data set is randomly divided into the training set and test set at the ratio of 8:2. In the training of rolling bearing fault diagnosis model based on Spark platform or Spark-GPU platform, the size of the training set that can be used should consider not only the hardware resource limitations of the cluster but also the model training efficiency. If a larger-scale rolling bearing data set is used, more worker nodes are required or the hardware configuration of each worker node is needed to be enhanced.

B. ANALYSIS OF FAULT DIAGNOSIS ACCURACY

In this experiment, for DataSet 1, DataSet 2, and DataSet 3, BPNN implemented with Spark (Spark-BPNN), QPSO-BPNN implemented with Spark (Spark-QPSO-BPNN), BPNN implemented with Spark-GPU (Spark-GPU-BPNN), and QPSO-BPNN implemented with Spark-GPU (Spark-GPU-QPSO-BPNN) are used for training and testing the rolling bearing fault diagnosis models, respectively. In the training of these fault diagnosis models, the key parameter settings of QPSO and BPNN are as follows.

- QPSO: The number of particles is set to 100, the shrinkage factor is set to 0.8, the max iterations is set to 50, and the error goal is set to 0.001.
- BPNN: The learning rate is set to 0.003, the momentum is set to 0.9, and the max iterations is set to 50.

The number of particles is one of the most important parameters of QPSO algorithm, too many particles will increase the computational cost, but too few particles will decrease the optimization effect. The setting of the shrinkage factor will affect the convergence speed of QPSO algorithm, if it is set too small, the convergence speed will be very slow; if it is set too large, the algorithm may fail to converge to an optimal solution. The learning rate is one of the most important parameters of BPNN, the setting of the learning rate will directly affect the convergence performance of BPNN, and it is usually between 0.001 and 0.01. The setting of the momentum will also affect the convergence speed of BPNN, and generally a larger value of momentum will increase the convergence speed.

Fig. 6 shows the diagnosis accuracies achieved using four different fault diagnosis methods and three different size data sets on the cluster described in Table. 1. As depicted in Fig. 6, the fault diagnosis accuracy achieved with Spark-QPSO-BPNN is 2.40% higher than that achieved with Spark-BPNN on average, and the fault diagnosis accuracy achieved with Spark-GPU-QPSO-BPNN is 2.41% higher than that achieved

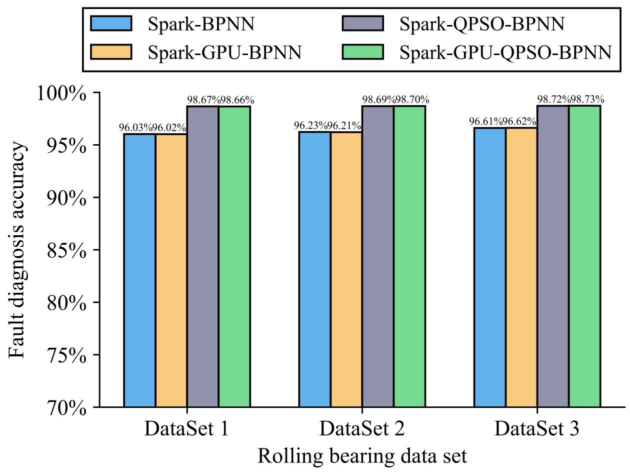

FIGURE 6. Diagnosis accuracies achieved using different fault diagnosis methods and different size data sets.

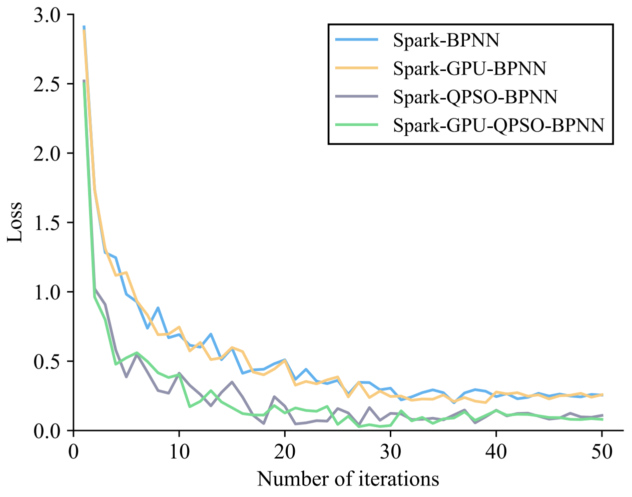

FIGURE 7. Loss curves of different fault diagnosis methods for dataSet 3.

with Spark-GPU-BPNN on average. The results show that QPSO algorithm can effectively optimize the initial weights and thresholds of BPNN, thereby obtaining a higher fault diagnosis accuracy.

It can be seen from Fig. 6 that the diagnosis accuracies achieved with Spark-GPU-QPSO-BPNN reach 98.66%, 98.70%, and 98.73% for DataSet 1, DataSet 2, and DataSet 3, respectively, which shows that the fault diagnosis accuracy is improved with the increase of data set size. This is because the fault features contained in the training samples become more and more with the increase of rolling bearing data set size, which helps to improve the fault diagnosis accuracy.

It can also be seen from Fig. 6 that the fault diagnosis accuracy achieved with Spark-BPNN and that achieved with Spark-GPU-BPNN are almost the same, and the fault diagnosis accuracy achieved with Spark-QPSO-BPNN and that achieved with Spark-GPU-QPSO-BPNN are also almost the same. The results show that the use of GPU will not affect the fault diagnosis accuracy of rolling bearing. The use of GPU in the proposed fault diagnosis method is mainly to improve the training efficiency and diagnosis efficiency of rolling bearing fault diagnosis model.

Fig. 7 presents the loss curves of four different fault diagnosis methods for DataSet 3. As shown in Fig. 7, the loss curves of the four methods all decrease rapidly at the first 10 iterations, then they decrease slowly with the increase of iterations, and they become stable gradually after the 40th iteration. The results show that the fault diagnosis models are well trained, the weights and thresholds of BPNN are continuously optimized during the training period, and the optimal weights and thresholds are obtained at the end of the training of the models. It can be found from Fig. 7 that the loss values of Spark-QPSO-BPNN and Spark-GPU-QPSO-BPNN are smaller than that of Spark-BPNN and Spark-GPU-BPNN, this is because the initial weights and thresholds of BPNN are effectively optimized by QPSO algorithm.

TABLE 4. Model training time and fault diagnosis time obtained under different size data sets.

Data Set	Model Training Time (minutes)			Fault Diagnosis Time (minutes)		
	Local-QPSO-BPNN	Spark-QPSO-BPNN	Spark-GPU-QPSO-BPNN	Local-QPSO-BPNN	Spark-QPSO-BPNN	Spark-GPU-QPSO-BPNN
DataSet 1	2495.82	122.27	7.70	25.41	2.54	0.23
DataSet 2	4939.88	237.37	14.75	47.13	4.48	0.38
DataSet 3	9498.34	447.87	26.69	92.96	8.48	0.65

C. PERFORMANCE ANALYSIS OF MODEL TRAINING AND FAULT DIAGNOSIS UNDER DIFFERENT SIZE DATA SETS

In order to analyze the performance of model training and fault diagnosis achieved with the proposed fault diagnosis method under different size data sets, for three different size data sets, Local-QPSO-BPNN, Spark-QPSO-BPNN, and Spark-GPU-QPSO-BPNN are used to train rolling bearing fault diagnosis models, and then the trained models are used for fault diagnosis. In this experiment, Local-QPSO-BPNN uses one CPU core of a single worker node to perform model training and fault diagnosis in local model, whereas Spark-QPSO-BPNN and Spark-GPU-QPSO-BPNN perform model training and fault diagnosis on the cluster with 4 worker nodes. To better analyze the performance of fault diagnosis achieved with the proposed rolling bearing fault diagnosis method on the massive data, all the data in each data set are diagnosed, namely the data of 8 GB, 16 GB, and 32 GB are diagnosed respectively. For three different size data sets, the time spent on model training and fault diagnosis using Local-QPSO-BPNN, Spark-QPSO-BPNN, and Spark-GPU-QPSO-BPNN, respectively, are shown in Table. 4.

Fig. 8 shows the speedups of Spark-GPU-QPSO-BPNN over Local-QPSO-BPNN under different size data sets. The speedup is the ratio of the model training time or fault diagnosis time achieved with Local-QPSO-BPNN to the model training time or fault diagnosis time achieved with Spark-GPU-QPSO-BPNN. As seen from Fig. 8, the proposed Spark-GPU-QPSO-BPNN achieves a significant performance improvement compared with Local-QPSO-BPNN.

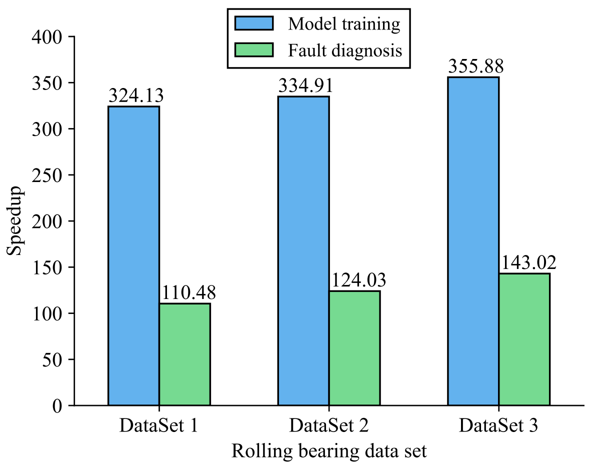

FIGURE 8. **Speedups of Spark-GPU-QPSO-BPNN over Local-QPSO-BPNN.**

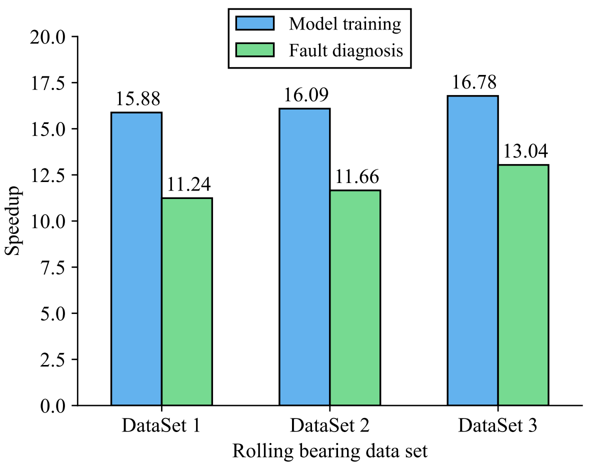

FIGURE 9. **Speedups of Spark-GPU-QPSO-BPNN over Spark-QPSO-BPNN.**

For DataSet 1, DataSet 2, and DataSet 3, Spark-GPU-QPSO-BPNN obtains the speedups of 324.13×, 334.91×, and 355.88× over Local-QPSO-BPNN for model training respectively, and Spark-GPU-QPSO-BPNN obtains the speedups of 110.48×, 124.03×, and 143.02× over Local-QPSO-BPNN for fault diagnosis respectively. This is mainly because Spark-GPU-QPSO-BPNN can fully utilize many-core GPUs of multiple worker nodes to efficiently perform model training and fault diagnosis in parallel on the Spark-GPU platform based on memory computing, which greatly improves the performance of model training and fault diagnosis under large-scale data sets. Moreover, the speedup obtained for model training is higher than that obtained for fault diagnosis, because Spark and GPU can give full play to their computational advantages in model training with a large number of iterative computations.

As can also be seen from Fig. 8, the speedups obtained for model training and fault diagnosis are gradually increased with the increase of data set size. This is because when Spark-GPU-QPSO-BPNN is used for model training and fault diagnosis on the cluster with 4 worker nodes, with the increase of data set size, the utilization of computing resources of GPU in each worker node is increased, and the parallel efficiencies of model training and fault diagnosis are also increased. Thus, the proposed fault diagnosis method is more suitable to deal with large-scale data sets.

Fig. 9 presents the speedups of Spark-GPU-QPSO-BPNN over Spark-QPSO-BPNN under different size data sets. The speedup is the ratio of the model training time or fault diagnosis time achieved with Spark-QPSO-BPNN to the model training time or fault diagnosis time achieved with Spark-GPU-QPSO-BPNN. As shown in Fig. 9, compared with Spark-QPSO-BPNN, the performance of model training and fault diagnosis achieved with Spark-GPU-QPSO-BPNN is significantly improved for different size data sets. For DataSet 1, DataSet 2, and DataSet 3, Spark-GPU-QPSO-BPNN obtains the speedups of 15.88×, 16.09×, and 16.78× over Spark-QPSO-BPNN for model training respectively, and Spark-GPU-QPSO-BPNN obtains the speedups of 11.24×, 11.66×, and 13.04× over Spark-QPSO-BPNN for fault diagnosis respectively. The results prove that the use of GPU can greatly improve the speeds of model training and fault diagnosis. This is mainly because most of the computations in BPNN are matrix operations, and many-core GPUs are more suitable for the parallel operations of large-scale matrices than multi-core CPUs.

TABLE 5. **Model training time and fault diagnosis time obtained under different size clusters.**

Cluster Size	Model Training Time (minutes)		Fault Diagnosis Time (minutes)	
	Spark-QPSO-BPNN	Spark-GPU-QPSO-BPNN	Spark-QPSO-BPNN	Spark-GPU-QPSO-BPNN
1 worker node	1603.38	97.15	30.95	1.62
2 worker nodes	862.03	51.68	16.48	1.26
3 worker nodes	585.17	35.53	11.33	0.91
4 worker nodes	447.87	26.69	8.48	0.65

D. PERFORMANCE ANALYSIS OF MODEL TRAINING AND FAULT DIAGNOSIS UNDER DIFFERENT SIZE CLUSTERS

In order to analyze the performance of model training and fault diagnosis of the proposed rolling bearing fault diagnosis method under different size clusters, Spark-QPSO-BPNN and Spark-GPU-QPSO-BPNN are adopted to perform model training and fault diagnosis respectively for DataSet 3 on the clusters with different numbers of worker nodes.

Table. 5 presents the model training time and fault diagnosis time obtained under different size clusters. As seen in Table. 5, as the number of worker nodes in the cluster increases, the model training time and fault diagnosis time achieved with Spark-QPSO-BPNN and Spark-GPU-QPSO-BPNN are gradually reduced. Compared with the cluster with a single worker node, on the clusters with 2, 3, and 4 worker nodes, the model training time achieved with Spark-GPU-QPSO-BPNN are reduced by 46.80%, 63.43%, and 72.53% respectively, and the fault diagnosis time achieved with Spark-GPU-QPSO-BPNN are reduced by 22.22%, 43.83%, and 59.88% respectively. The results show that the increase of cluster size can effectively improve the performance of model training and fault

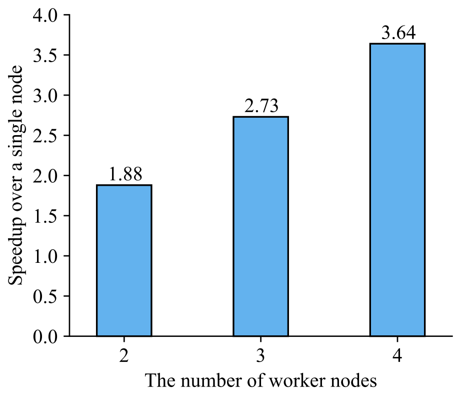

FIGURE 10. Speedups achieved with Spark-GPU-QPSO-BPNN for model training.

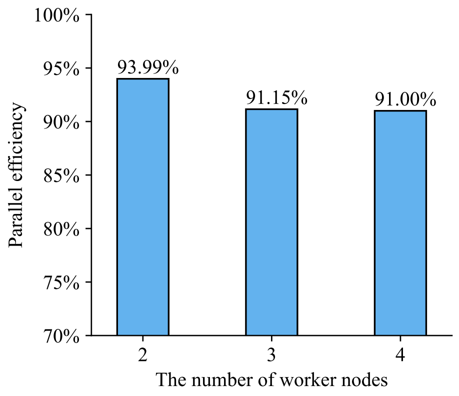

FIGURE 11. Parallel efficiencies achieved with Spark-GPU-QPSO-BPNN for model training.

diagnosis for the proposed rolling bearing fault diagnosis method. Moreover, compared with Spark-QPSO-BPNN, the model training time and fault diagnosis time achieved with Spark-GPU-QPSO-BPNN are significantly reduced under different size clusters, which once again proves that the use of GPU can significantly improve the speeds of model training and fault diagnosis.

Fig. 10 shows the speedups achieved with Spark-GPU-QPSO-BPNN for model training. The speedup is the ratio of the model training time achieved with a single worker node to the model training time achieved with multiple worker nodes. As shown in Fig. 10, the speedups achieved with Spark-GPU-QPSO-BPNN are increased with the increase of the number of worker nodes in the cluster. On the clusters with 2, 3, and 4 worker nodes, the speedups achieved with Spark-GPU-QPSO-BPNN are 1.88×, 2.73×, and 3.64× respectively, which shows that the QPSO-BPNN model is well distributed and parallelized on Spark-GPU platform.

Fig. 11 presents the parallel efficiencies achieved with Spark-GPU-QPSO-BPNN for model training. The parallel efficiency is the ratio of the speedup obtained for model training to the number of worker nodes in the cluster. As shown in Fig. 11, when the numbers of worker nodes in the cluster are 2, 3, and 4, the parallel efficiencies achieved with Spark-GPU-QPSO-BPNN reach 93.99%, 91.15%, and 91.00% respectively, which shows that the computing resources of the

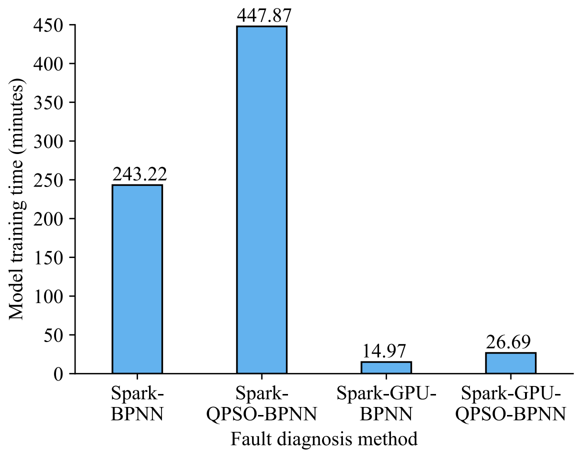

FIGURE 12. Comparison of the model training time achieved using QPSO and without QPSO.

Spark-GPU platform have been fully utilized. However, as the number of worker nodes increases, the parallel efficiency is gradually decreased. This is because the increase of cluster size will lead to the increases of communication overhead and task scheduling overhead between nodes, which will affect the performance of model training.

E. ANALYSIS OF THE IMPACT OF QPSO ON THE PERFORMANCE OF MODEL TRAINING

In order to analyze the impact of QPSO on the performance of model training, Spark-BPNN, Spark-QPSO-BPNN, Spark-GPU-BPNN, and Spark-GPU-QPSO-BPNN are used for training and testing the fault diagnosis models respectively for DataSet 3 on the cluster with 4 worker nodes.

Fig. 12 gives the comparison of the model training time achieved using QPSO and without QPSO. The model training time achieved with Spark-QPSO-BPNN is increased by 84.14% than that achieved with Spark-BPNN, and the model training time achieved with Spark-GPU-QPSO-BPNN is increased by 78.29% than that achieved with Spark-GPU-BPNN. The main reason for the increase of model training time is that QPSO algorithm is used to optimize the initial weights and thresholds of BPNN in Spark-QPSO-BPNN and Spark-GPU-QPSO-BPNN, which requires more computational cost than the weights and thresholds of BPNN are randomly initialized in Spark-BPNN and Spark-GPU-BPNN. Although it takes a lot of time to optimize the initial weights and thresholds of BPNN, after obtaining the optimal initial weights and thresholds, Spark-QPSO-BPNN and Spark-GPU-QPSO-BPNN can converge to the global optimal weights and thresholds at a faster speed than Spark-BPNN and Spark-GPU-BPNN. Although using QPSO algorithm to optimize the initial weights and thresholds of BPNN will affect the model training efficiency, it can significantly improve the fault diagnosis accuracy. As shown in Fig. 6, the fault diagnosis accuracy achieved with Spark-GPU-QPSO-BPNN is 2.11% higher than that achieved with Spark-GPU-BPNN for DataSet 3.

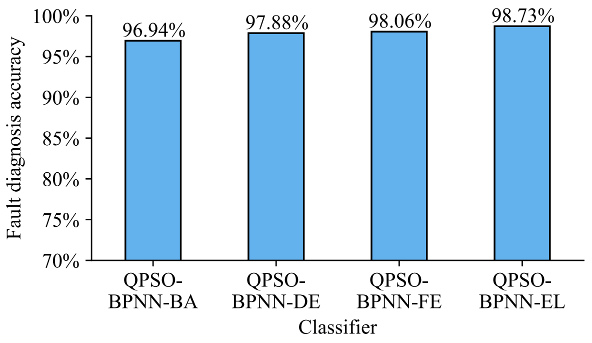

FIGURE 13. Fault diagnosis accuracies achieved with different classifiers.

F. ANALYSIS OF COMBINATION EFFECT OF MULTIPLE QPSO-BPNN MODELS

In order to verify the effectiveness of the proposed combination strategy of multiple QPSO-BPNN models based on ensemble learning, four different classifiers are adopted for training and testing the fault diagnosis models respectively for DataSet 3, including the QPSO-BPNN model of base end (QPSO-BPNN-BA), the QPSO-BPNN model of drive end (QPSO-BPNN-DE), the QPSO-BPNN model of fan end (QPSO-BPNN-FE), and the ensemble classifier composed of the above three different QPSO-BPNN models based on ensemble learning (QPSO-BPNN-EL).

Fig. 13 presents the fault diagnosis accuracies achieved with different classifiers. Compared with QPSO-BPNN-BA, QPSO-BPNN-DE, and QPSO-BPNN-FE, the fault diagnosis accuracy achieved with QPSO-BPNN-EL is increased by 1.79%, 0.85%, and 0.67% respectively. The results show that the ensemble classifier can effectively improve the fault diagnosis accuracy.

To further demonstrate the effectiveness of the proposed combination strategy, a sample whose real state is the inner race fault but that is misdiagnosed by one basic classifier is selected. Table. 6 presents the diagnosis results of three basic classifiers and one ensemble classifier for the sample. As seen in Table. 6, the diagnosis results achieved with QPSO-BPNN-BA, QPSO-BPNN-DE, and QPSO-BPNN-FE are the inner race fault, inner race fault, and ball fault respectively, and the diagnosis result achieved with QPSO-BPNN-EL is the inner race fault. The results show that when the diagnosis results of different basic classifiers are inconsistent for a sample, the ensemble classifier can obtain the best diagnosis result according to the output results of each basic classifier.

G. COMPARISON WITH OTHER INTELLIGENT OPTIMIZATION ALGORITHMS

In order to better evaluate the optimization effect of QPSO algorithm, GA [34], APSO algorithm [38], and SPSO algorithm [39] are also adopted to optimize the initial weights and thresholds of BPNN. Spark-GPU-GA-BPNN, Spark-GPU-APSO-BPNN, Spark-GPU-SPSO-BPNN, and Spark-GPU-QPSO-BPNN are respectively used to perform model training on the cluster with 4 worker nodes for DataSet 3. During the training period, the max iterations of GA, APSO, SPSO, and QPSO are set to 50, and the other key parameter settings are as follows.

- GA: The size of population is set to 100, the mutation probability is set to 0.2, and the crossover probability is set to 0.5.
- APSO: The number of particles is set to 100 and two acceleration constants are set to 1.4945.
- SPSO: The number of particles is set to 100, two acceleration constants are set to 2, the initial weight value is set to 0.9, and the final weight value is set to 0.4.
- QPSO: See Section IV-B.

TABLE 6. Diagnosis results of different classifiers for a sample.

Classifier	Output Probability of Each Class				Fault Diagnosis Result
	Normal State	Inner Race Fault	Ball Fault	Outer Race Fault	
QPSO-BPNN-BA	0.1285	0.5759	0.2404	0.0552	Inner race fault
QPSO-BPNN-DE	0.0689	0.5892	0.3061	0.0358	Inner race fault
QPSO-BPNN-FE	0.2045	0.3259	0.3385	0.1311	Ball fault
QPSO-BPNN-EL	0.1342	0.4969	0.2951	0.0741	Inner race fault

TABLE 7. Comparison of different intelligent optimization algorithms.

Fault Diagnosis Method	Fault Diagnosis Accuracy	Model Training Time (minutes)
Spark-GPU-GA-BPNN	97.08%	46.08
Spark-GPU-APSO-BPNN	98.47%	29.93
Spark-GPU-SPSO-BPNN	98.15%	26.20
Spark-GPU-QPSO-BPNN	98.73%	26.69

Table. 7 presents the fault diagnosis accuracies and model training time of different fault diagnosis methods. The fault diagnosis accuracy achieved with Spark-GPU-QPSO-BPNN is 1.65%, 0.26% and 0.58% higher than that achieved with Spark-GPU-GA-BPNN, Spark-GPU-APSO-BPNN, and Spark-GPU-SPSO-BPNN, respectively. This is mainly because QPSO algorithm introduces the average best position of quantum particle swarm, which has better randomness and stronger global optimization ability than the other three algorithms when optimizing the initial weights and thresholds of BPNN.

As seen in Table. 7, the model training speed of Spark-GPU-QPSO-BPNN is 1.73× and 1.12× as fast as that of Spark-GPU-GA-BPNN and Spark-GPU-APSO-BPNN, respectively. This is mainly because QPSO algorithm removes the velocity attribute of the particle swarm, which can greatly reduce the computational cost of optimizing the initial weights and thresholds of BPNN. In addition, the model training speed of Spark-GPU-QPSO-BPNN and Spark-GPU-SPSO-BPNN is very close. The main reason is that SPSO algorithm also can greatly reduce the computational cost by shrinking the search space.

H. COMPARISON WITH OTHER FAULT DIAGNOSIS METHODS

To further verify the effectiveness of the proposed rolling bearing fault diagnosis method, the following four different

TABLE 8. Network structures and hyper-parameter settings of different fault diagnosis methods based on deep learning.

Rolling Bearing Fault Diagnosis Method	Number of Convolutional Layers	Number of Pooling Layers	Number of Fully Connected Layers	Batch Size	Learning Rate	Momentum	Number of Epochs
Spark-GPU-AlexNet	5	3	3	128	0.008	0.9	50
Spark-GPU-VGG-19	16	6	3	128	0.005	0.9	50
Spark-GPU-ResNet-18	20	2	1	64	0.003	0.9	50

TABLE 9. Comparison of different rolling bearing fault diagnosis methods based on Spark-GPU platform.

Fault Diagnosis Method	Fault Diagnosis Accuracy	Model Training Time (minutes)	Fault Diagnosis Time (minutes)
Spark-GPU-AlexNet	99.84%	144.35	5.17
Spark-GPU-VGG-19	99.89%	1540.05	18.83
Spark-GPU-ResNet-18	99.92%	503.04	7.74
Spark-GPU-QPSO-BPNN	98.73%	26.69	0.65

methods are used for model training and fault diagnosis on the cluster with 4 worker nodes: AlexNet [49] implemented with Spark-GPU (Spark-GPU-AlexNet), VGG-19 [50] implemented with Spark-GPU (Spark-GPU-VGG-19), ResNet-18 [51] implemented with Spark-GPU (Spark-GPU-ResNet-18), and the proposed Spark-GPU-QPSO-BPNN. For Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18, the original vibration data are converted into 64×64 pixel gray-scale images, and a gray-scale image data set with the same size as DataSet 3 is obtained. The data set is divided into training set and test set at the ratio of 8:2. For Spark-GPU-QPSO-BPNN, DataSet 3 is used as the experimental data set and divided into training set and test set at the ratio of 8:2. Moreover, Spark-GPU-QPSO-BPNN diagnoses all the data in DataSet 3, and the three deep learning methods diagnose all the data in the gray-scale image data set.

The network structures and hyper-parameter settings of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 are listed in Table. 8. Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 contain 11, 25, and 23 neural network layers, respectively. The detailed network structures of AlexNet, VGG-19, and ResNet-18 can be found in [49], [50], and [51]. The batch sizes of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 are set to 128, 128, and 64 respectively, which can achieve a better fault diagnosis accuracy within the limit of the available GPU memory. The learning rate of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 are set to 0.008, 0.005, and 0.003 respectively, which can provide a better convergence performance and avoid fluctuations in model training. The momentum of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 are all set to 0.9, which can increase the convergence speed. The number of epochs of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 are all set to 50, which can not only ensure a higher fault diagnosis accuracy, but also prevent the model training time from being too long.

Table. 9 presents the diagnosis accuracies, model training time, and fault diagnosis time of four different rolling bearing fault diagnosis methods based on Spark-GPU platform. The fault diagnosis accuracy achieved with Spark-GPU-QPSO-BPNN is 1.11%, 1.16%, and 1.19% lower than that achieved with Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18, respectively. However, the model training speed of Spark-GPU-QPSO-BPNN is 4.41×, 56.70×, and 17.85× faster than that of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 respectively, and the fault diagnosis speed of Spark-GPU-QPSO-BPNN is 6.95×, 27.97×, and 10.91× faster than that of Spark-GPU-AlexNet, Spark-GPU-VGG-19, and Spark-GPU-ResNet-18 respectively. Compared with AlexNet, VGG-19, and ResNet-18, QPSO-BPNN has a simpler network structure and fewer parameters, which can achieve higher model training efficiency and fault diagnosis efficiency. Therefore, the proposed rolling bearing fault diagnosis method not only can efficiently perform model training and fault diagnosis on massive rolling bearing vibration data, but also has good diagnosis accuracy.

V. CONCLUSION

To perform fast and accurate rolling bearing fault diagnosis in the big data environment, a rolling bearing fault diagnosis method based on parallel QPSO-BPNN under Spark-GPU platform is proposed. According to the idea of data parallelism, the distributed parallelization of QPSO-BPNN model is effectively realized on Spark-GPU platform, which significantly improves the performance of model training and fault diagnosis under large-scale rolling bearing data sets. In the distributed parallel training of QPSO-BPNN model, the master node collects the local parameters of each worker node and updates the global parameters according to the weights, and the updated global parameters are synchronized to each worker node, which effectively improves the convergence speed of the model. The combination strategy based on ensemble learning is adopted, and the output results of QPSO-BPNN models respectively corresponding to the base end, drive end, and fan end of rolling bearing are combined according to the weighted voting method to obtain the best fault diagnosis result of a sample. The effectiveness of the proposed fault diagnosis method is verified through experiments. The results illustrate that the proposed method can not only make full use of the computing resources of a Spark-GPU platform to efficiently perform model training and fault diagnosis but also obtain a higher fault diagnosis accuracy for the massive rolling bearing vibration data.

In the actual industrial production environments, the sensors deployed on rolling bearing can continuously collect vibration data. Facing a large amount of rolling bearing vibration data collected in real time, the rapid and accurate online fault diagnosis can effectively ensure the safe operation of mechanical equipment and reduce the maintenance cost. In future work, an online fault diagnosis method of rolling bearing based on parallel QPSO-BPNN in the big data environment will be explored.

REFERENCES

[1] R. Liu, B. Yang, E. Zio, and X. Chen, "Artificial intelligence for fault diagnosis of rotating machinery: A review," *Mech. Syst. Signal Process.*, vol. 108, pp. 33–47, Aug. 2018.

[2] B. Pang, G. Tang, and T. Tian, "Enhanced singular spectrum decomposition and its application to rolling bearing fault diagnosis," *IEEE Access*, vol. 7, pp. 87769–87782, 2019.

[3] X. Huang, G. Wen, L. Liang, Z. Zhang, and Y. Tan, "Frequency phase space empirical wavelet transform for rolling bearings fault diagnosis," *IEEE Access*, vol. 7, pp. 86306–86318, 2019.

[4] Z. Ma, F. Lu, S. Liu, and X. Li, "An adaptive generalized demodulation method for multimedia spectrum analysis is applied in rolling bearing fault diagnosis," *IEEE Access*, vol. 8, pp. 20687–20699, 2020.

[5] W. Liu, W. Chen, and Z. Zhang, "A novel fault diagnosis approach for rolling bearing based on high-order synchrosqueezing transform and detrended fluctuation analysis," *IEEE Access*, vol. 8, pp. 12533–12541, 2020.

[6] T. Jiang, J. Wang, C. Shen, X. Jiang, and Z. Zhu, "Multi-bandwidth mode manifold for fault diagnosis of rolling bearings," *IEEE Access*, vol. 7, pp. 179620–179633, 2019.

[7] B. Chen, B. Shen, F. Chen, H. Tian, W. Xiao, F. Zhang, and C. Zhao, "Fault diagnosis method based on integration of RSSD and wavelet transform to rolling bearing," *Measurement*, vol. 131, pp. 400–411, Jan. 2019.

[8] N. Zhang, L. Wu, J. Yang, and Y. Guan, "Naive bayes bearing fault diagnosis based on enhanced independence of data," *Sensors*, vol. 18, no. 2, p. 463, Feb. 2018.

[9] X. Li, Y. Yang, H. Pan, J. Cheng, and J. Cheng, "A novel deep stacking least squares support vector machine for rolling bearing fault diagnosis," *Comput. Ind.*, vol. 110, pp. 36–47, Sep. 2019.

[10] X. Qin, J. Guo, X. Dong, and Y. Guo, "The fault diagnosis of rolling bearing based on variational mode decomposition and iterative random forest," *Shock Vib.*, vol. 2020, Feb. 2020, Art. no. 1576150.

[11] J. Li, X. Yao, X. Wang, Q. Yu, and Y. Zhang, "Multiscale local features learning based on BP neural network for rolling bearing intelligent fault diagnosis," *Measurement*, vol. 153, Mar. 2020, Art. no. 107419.

[12] L. Eren, T. Ince, and S. Kiranyaz, "A generic intelligent bearing fault diagnosis system using compact adaptive 1D CNN classifier," *J. Signal Process. Syst.*, vol. 91, no. 2, pp. 179–189, Feb. 2019.

[13] G. Li, C. Deng, J. Wu, Z. Chen, and X. Xu, "Rolling bearing fault diagnosis based on wavelet packet transform and convolutional neural network," *Appl. Sci.*, vol. 10, no. 3, p. 770, Jan. 2020.

[14] M. Qiao, S. Yan, X. Tang, and C. Xu, "Deep convolutional and LSTM recurrent neural networks for rolling bearing fault diagnosis under strong noises and variable loads," *IEEE Access*, vol. 8, pp. 66257–66269, 2020.

[15] S. Gao, L. Xu, Y. Zhang, and Z. Pei, "Rolling bearing fault diagnosis based on intelligent optimized self-adaptive deep belief network," *Meas. Sci. Technol.*, vol. 31, no. 5, May 2020, Art. no. 055009.

[16] F. Zhou, S. Yang, H. Fujita, D. Chen, and C. Wen, "Deep learning fault diagnosis method based on global optimization GAN for unbalanced data," *Knowl.-Based Syst.*, vol. 187, Jan. 2020, Art. no. 104837.

[17] M. Zhao, M. Kang, B. Tang, and M. Pecht, "Multiple wavelet coefficients fusion in deep residual networks for fault diagnosis," *IEEE Trans. Ind. Electron.*, vol. 66, no. 6, pp. 4696–4706, Jun. 2019.

[18] J. Shao, Z. Huang, and J. Zhu, "Transfer learning method based on adversarial domain adaption for bearing fault diagnosis," *IEEE Access*, vol. 8, pp. 119421–119430, 2020.

[19] Y. Xu, Y. Sun, J. Wan, X. Liu, and Z. Song, "Industrial big data for fault diagnosis: Taxonomy, review, and applications," *IEEE Access*, vol. 5, pp. 17368–17380, 2017.

[20] H. Yan, J. Wan, C. Zhang, S. Tang, Q. Hua, and Z. Wang, "Industrial big data analytics for prediction of remaining useful life based on deep learning," *IEEE Access*, vol. 6, pp. 17190–17197, 2018.

[21] G. Xian, "Parallel machine learning algorithm using fine-grained-mode spark on a mesos big data cloud computing software framework for mobile robotic intelligent fault recognition," *IEEE Access*, vol. 8, pp. 131885–131900, 2020.

[22] H. Miao, H. Zhang, M. Chen, B. Qi, and J. Li, "Two-level fault diagnosis of SF6 electrical equipment based on big data analysis," *Big Data Cognit. Comput.*, vol. 3, no. 1, p. 4, Jan. 2019.

[23] W. Shi, Y. Zhu, T. Huang, G. Sheng, Y. Lian, G. Wang, and Y. Chen, "An integrated data preprocessing framework based on apache spark for fault diagnosis of power grid equipment," *J. Signal Process. Syst.*, vol. 86, nos. 2–3, pp. 221–236, Mar. 2017.

[24] M. B. Imani, M. Heydarzadeh, L. Khan, and M. Nourani, "A scalable spark-based fault diagnosis platform for gearbox fault diagnosis in wind farms," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, San Diego, CA, USA, Aug. 2017, pp. 100–107.

[25] W. Yu, T. Dillon, F. Mostafa, W. Rahayu, and Y. Liu, "A global manufacturing big data ecosystem for fault detection in predictive maintenance," *IEEE Trans. Ind. Informat.*, vol. 16, no. 1, pp. 183–192, Jan. 2020.

[26] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[27] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[28] M. M. Rathore, H. Son, A. Ahmad, A. Paul, and G. Jeon, "Real-time big data stream processing using GPU with spark over Hadoop ecosystem," *Int. J. Parallel Program.*, vol. 46, no. 3, pp. 630–646, Jun. 2017.

[29] R. N. Boubela, K. Kalcher, W. Huf, C. Našel, and E. Moser, "Big data approaches for the analysis of large-scale fMRI data using Apache Spark and GPU processing: A demonstration on resting-state fMRI data from the human connectome project," *Frontiers Neurosci.*, vol. 9, p. 492, Jan. 2016.

[30] D. Lunga, J. Gerrand, L. Yang, C. Layton, and R. Stewart, "Apache spark accelerated deep learning inference for large scale satellite image analytics," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 13, no. 2, pp. 271–283, Jan. 2020.

[31] L. Wan, H. Li, Y. Chen, and C. Li, "Rolling bearing fault prediction method based on QPSO-BP neural network and dempster–shafer evidence theory," *Energies*, vol. 13, no. 5, p. 1094, Mar. 2020.

[32] L. Wan, G. Zhang, H. Li, and C. Li, "A novel bearing fault diagnosis method using spark-based parallel ACO-K-Means clustering algorithm," *IEEE Access*, vol. 9, pp. 28753–28768, 2021.

[33] J. Li, J.-H. Cheng, J.-Y. Shi, and F. Huang, "Brief introduction of back propagation (BP) neural network algorithm and its improvement," in *Advances in Computer Science and Information Engineering*. Berlin, Germany: Springer, 2012, pp. 553–558.

[34] Y. Wen, M. Jia, and C. Luo, "Study of fault diagnosis for rolling bearing based on GA-BP algorithm," in *Proc. 2nd Int. Conf. Autom., Mech. Control Comput. Eng. (AMCCE)*, Beijing, China, 2017, pp. 776–781.

[35] J. Shi, X. Wu, J. Zhou, and S. Wang, "BP neural network based bearing fault diagnosis with differential evolution & EEMD denoise," in *Proc. 9th Int. Conf. Model., Identificat. Control (ICMIC)*, Kunming, China, Jul. 2017, pp. 1038–1043.

[36] H. Yuan, X. Wang, X. Sun, and Z. Ju, "Compressive sensing-based feature extraction for bearing fault diagnosis using a heuristic neural network," *Meas. Sci. Technol.*, vol. 28, no. 6, Jun. 2017, Art. no. 065018.

[37] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization," *AIAA J.*, vol. 41, no. 8, pp. 1583–1589, 2003.

[38] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 6, pp. 1362–1381, Dec. 2009.

[39] T. M. Khalil and A. V. Gorpinich, "Selective particle swarm optimization," *Int. J. Multidisciplinary Sci. Eng.*, vol. 3, no. 4, pp. 1–4, Apr. 2012.

[40] Y. Zhang, S. Wang, and G. Ji, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Math. Probl. Eng.*, vol. 2015, Oct. 2015, Art. no. 931256.

[41] RAPIDS Development Team. (2018). *RAPIDS: Collection of Libraries for End to End GPU Data Science*. Accessed: Jun. 22, 2020. [Online]. Available: https://rapids.ai

[42] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, Broomfield, CO, USA, 2014, pp. 583–598.

[43] D. Stathakis, "How many hidden layers and nodes?" *Int. J. Remote Sens.*, vol. 30, no. 8, pp. 2133–2147, Apr. 2009.

[44] S. N. Omkar, R. Khandelwal, T. V. S. Ananth, G. N. Naik, and S. Gopalakrishnan, "Quantum behaved particle swarm optimization (QPSO) for multi-objective design optimization of composite structures," *Expert Syst. Appl.*, vol. 36, no. 8, pp. 11312–11322, Oct. 2009.

[45] K. Sentz and S. Ferson, *Combination Evidence Dempster-Shafer Theory*. Albuquerque, NM, USA: Sandia National Laboratory, 2002, pp. 8–15.

[46] C. Zhang and Y. Ma, *Ensemble Machine Learning: Methods and Applications*. New York, NY, USA: Springer, 2012, pp. 1–30.

[47] Case Western Reserve University Bearing Data Center. (2012). *Seeded Fault Test Data*. Accessed: Jun. 18, 2020. [Online]. Available: https://csegroups.case.edu/bearingdatacenter/home

[48] U. Yun, G. Lee, and E. Yoon, "Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields," *Inf. Sci.*, vol. 494, pp. 37–59, Aug. 2019.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, 2015, pp. 1–14.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

LANJUN WAN was born in Hunan, China, in 1982. He received the B.S. and M.S. degrees in computer science and technology from the Hunan University of Technology, Zhuzhou, China, in 2005 and 2009, respectively, and the Ph.D. degree in circuits and systems from Hunan University, Changsha, China, in 2016. He is currently an Assistant Professor with the School of Computer Science, Hunan University of Technology. His research interests include industrial big data analysis, industrial equipment fault diagnosis, high-performance computing, and parallel computing.

HONGYANG LI was born in Heilongjiang, China, in 1995. He received the B.S. degree in electronic science and technology from the Tianjin University of Technology, Tianjin, China, in 2017. He is currently pursuing the M.S. degree in computer science and technology with the Hunan University of Technology, Zhuzhou, China. His research interests include industrial big data analysis and industrial equipment fault diagnosis.

GEN ZHANG was born in Anhui, China, in 1995. He received the B.S. degree in network engineering from West Anhui University, Luan, China, in 2019. He is currently pursuing the M.S. degree in computer science and technology with the Hunan University of Technology, Zhuzhou, China. His research interests include industrial big data analysis and industrial equipment fault diagnosis.

CHANGYUN LI was born in Hunan, China, in 1972. He received the Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2007. He is currently a Full Professor of computer science and the Dean of the Graduate School, Hunan University of Technology, Zhuzhou, China. His major research interests include industrial big data analysis, industrial equipment fault diagnosis, and intelligent information perception and processing technology.

JUNFENG MAN was born in Heilongjiang, China, in 1976. He received the Ph.D. degree in computer science and technology from Central South University, Changsha, China, in 2010. He is currently a Full Professor with the School of Computer Science, Hunan University of Technology, Zhuzhou, China. His major research interests include industrial big data analysis, industrial equipment fault diagnosis, industry equipment health management, and industrial Internet.

MANSHENG XIAO was born in Hunan, China, in 1968. He received the M.S. degree in computer science and technology from Xi'an Jiaotong University, Xi'an, China, in 2005. He is currently a Full Professor with the School of Computer Science, Hunan University of Technology, Zhuzhou, China. His major research interests include industrial big data analysis, intelligent information processing, pattern recognition, and image processing.