Mechanical Systems and Signal Processing 168 (2022) 108653

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

The emerging graph neural networks for intelligent fault diagnostics and prognostics: A guideline and a benchmark study

Tianfu Li, Zheng Zhou, Sinan Li, Chuang Sun *, Ruqiang Yan, Xuefeng Chen

School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China

ARTICLE INFO

Communicated by Xiaosheng SI

Keywords:
Prognostics and health management
Graph neural networks
Intelligent fault diagnostics and prognostics
Practical guideline
Benchmark results

ABSTRACT

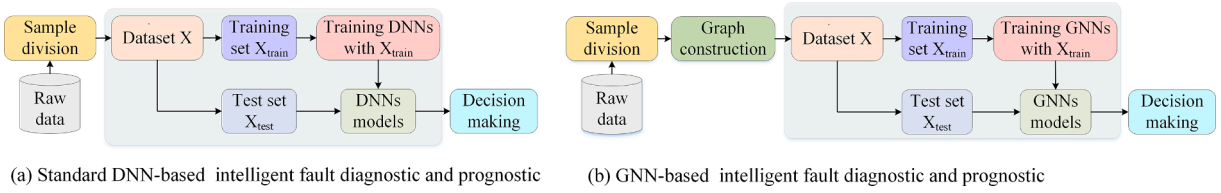
Deep learning (DL)-based methods have advanced the field of Prognostics and Health Management (PHM) in recent years, because of their powerful feature representation ability. The data in PHM are typically regular data represented in the Euclidean space. Nevertheless, there are an increasing number of applications that consider the relationships and interdependencies of data and represent the data in the form of graphs. Such kind of irregular data in non-Euclidean space pose a huge challenge to the existing DL-based methods, making some important operations (e.g., convolutions) easily applied to Euclidean space but difficult to model graph data in non-Euclidean space. Recently, graph neural networks (GNNs), as the emerging neural networks, have been utilized to model and analyze the graph data. However, there still lacks a guideline on leveraging GNNs for realizing intelligent fault diagnostics and prognostics. To fill this research gap, a practical guideline is proposed in this paper, and a novel intelligent fault diagnostics and prognostics framework based on GNN is established to illustrate how the proposed guideline works. In this framework, three types of graph construction methods are provided, and seven kinds of graph convolutional networks (GCNs) with four different graph pooling methods are investigated. To afford benchmark results for helping further study, a comprehensive evaluation of these models is performed on eight datasets, including six fault diagnosis datasets and two prognosis datasets. Finally, four issues related to the performance of GCNs are discussed and potential research directions are provided. The code library is available at: https://github.com/HazeDT/PHMGNNBenchmark.

1. Introduction

Prognostics and Health Management (PHM) with the goal to diagnose the health status of equipment and predict the occurrence of failures through data monitoring and analysis, thereby greatly improving the efficiency of condition-based maintenance [1,2]. Intelligent diagnostic and prognostic, as two key components of the PHM system, have been widely applied to monitoring rotating machinery [3–5], such as wind turbine, helicopter, high-speed train, and aero-engine, etc. Traditional intelligent diagnostic and prognostic methods mainly contain two steps, that is, feature extraction by using signal processing approaches [6,7] and fault classification or regression by using machine learning methods [8,9]. However, faced with the industrial big data, the process of using traditional methods to extract features is very time-consuming and laborious. Besides, the effectiveness of the extracted features greatly depends on the expert knowledge, which limits the application of intelligent diagnostic and prognostic methods in industry.

* Corresponding author.
E-mail address: ch.sun@xjtu.edu.cn (C. Sun).

https://doi.org/10.1016/j.ymssp.2021.108653
Received 21 May 2021; Received in revised form 30 October 2021; Accepted 14 November 2021
Available online 1 December 2021



(a) Standard DNN-based intelligent fault diagnostic and prognostic (b) GNN-based intelligent fault diagnostic and prognostic

Fig. 1. The framework of DNN-based methods and GNN-based methods.

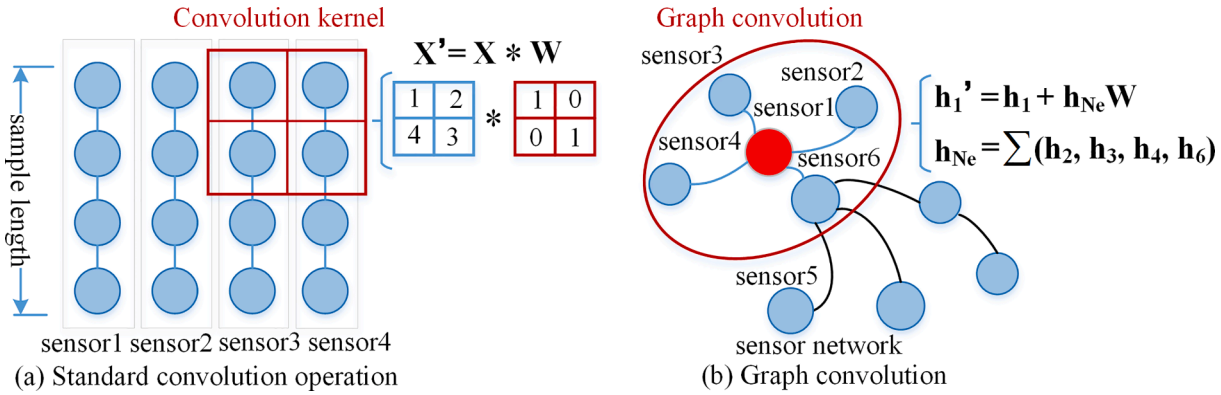


Fig. 2. An illustration of convolution operation and graph convolution operation. (a) The results of the standard convolution operation are the sum of the point-wise multiplications of a subset of the signal **X** with a learned convolution filter **W**. (b) The results of graph convolution operation could be the sum of the node feature itself with the product of the features of adjacent nodes $\mathbf{h_{Ne}}$ and the learnable parameter **W**.

Benefiting from the prosperity of deep learning (DL) theory and computational resources, intelligent diagnostic and prognostic techniques have made great progress in feature extraction [10–12], which use efficient neural network architectures to automatically extract features from the raw signals. Nowadays, many powerful deep neural networks (DNNs) have been developed, such as convolutional neural networks (CNNs) [13–15], auto-encoders (AEs) [16–18], deep belief networks (DBNs) [19–21], and recurrent neural networks (RNNs) [22–24]. These models have also been successfully applied in PHM. For example, Zhu et al. [25] proposed a transferable remaining useful life prediction approach, where the fault occurrence time was located by a hidden Markov model and the distribution of discrepancy was aligned by a multi-layer perceptron. Yu et al. [26] proposed a three-stage semi-supervised CNN for rolling bearing fault diagnostics, including the data augmentation stage, the K-means clustering stage, and the feature distribution alignment stage. Shao et al. [27] adopted the Gaussian wavelet function as the activation and proposed a deep wavelet auto-encoder for achieving fault diagnostics of electric locomotive bearings.

Currently, the framework of standard DNN-based intelligent fault diagnostics and prognostics generally includes four steps, that is, data collection, model construction, model training, and decision making, as shown in Fig. 1(a) [28]. In data collection, the raw data is collected and divided into subsamples. Then, a suitable DNN model is constructed and trained with the obtained training set. Since the DNN model is well trained, the decision making on the category of a new sample can be conducted based on this model. Although these DNN-based methods can effectively capture hidden features of regular data (e.g., images, and time series.), most of them ignore the interdependencies between data or various physical measurements of multiple sensors. For example, as shown in Fig. 2(a), when the standard convolution operation convolves multi-sensor measurements, it only takes the weighted sum of the sensor measurement and the corresponding convolution kernel without considering the interaction between the sensors.

To address this issue, there are a growing number of applications that consider the interdependencies between data and display the data in the form of irregular graphs [29–32]. In graph data, the relationships between nodes are reflected on the connecting edges, and the edge weight reflects the strength of the relationship. As depicted in Fig. 2, the collected time series by multiple sensors are the commonly acquired regular data, and the sensor network constructed by multiple sensors is regarded as irregular data. Compared with the irregular graph, it can be found that the number of neighbors of each node on the irregular graph could be different, while the number of neighbors of the regular data is constant. However, the complexity of graph data brings a critical challenge to the standard DNN-based methods, which makes some important operations (e.g., convolutions) easily applied to the Euclidean domain but difficult to model graph data in non-Euclidean space.

Graph neural networks (GNNs) [33], as the emerging neural networks, are designed to model the graph data. Motivated by CNNs, RNNs and AEs in DL, new concepts and definitions have been extended on complex graph data and spawned the corresponding graph convolutional neural networks (GCNs) [34], graph recurrent neural networks (GRNNs) [35] and graph auto-encoders (GAEs) [36]. These GNNs have been successfully implemented in many fields [37], such as chemistry [38], commonsense reasoning [39], natural language processing [40], social network [41], and traffic flow forecasting [42]. In graph domain, there are three tasks that can be modeled by GNNs [43], including: 1) node classification, where GNNs are used to obtain the representation of each node and conduct

node classification; 2) graph classification or regression, where GNNs are utilized to obtain the representation of the entire graph and then for graph classification or regression; 3) link prediction, where GNNs are applied to mine the relationship between each node and predict the missing edges.

Recently, due to the ability of GNNs to model the interdependencies between data and embed them into the extracted features, they have gradually been applied to PHM by researchers [44–46]. For instance, Zhang et al. [47] transformed the collected acoustic signals into graphs and applied GCN to model the graphs, thus realizing fault diagnosis of roller bearings. Yu et al. [48] adopted a wavelet packet to decompose the vibration signals of the wind turbine gearbox and obtained the graph dataset, and then fault classification was achieved by the proposed fast deep GCN. Li et al. [49] transformed the vibration signals of rolling bearings into horizontal visibility graphs, and then a GNN was applied to model the graph data and realize fault classification. From the literature review, it can be found that the main differences between DNN-based methods and GNN-based methods lie in the graph construction and the designed models, as shown in Fig. 1(b). Therefore, how to transform the raw data into graphs and design an appropriate network are two key issues faced in GNN-based methods. However, there still lacks a practical guideline on leveraging GNNs to realize intelligent fault diagnostics and prognostics. In addition, a benchmark study on existing methods is also necessary, which can provide guidance to researchers for further study.

To fill this research gap, a practical guideline for establishing a novel intelligent fault diagnostics and prognostics framework based on GNNs is illustrated. In this framework, three types of graph construction methods are provided, and seven kinds of GNNs with four different graph pooling methods are investigated. A benchmark study of these GNNs is carried out on six fault diagnosis datasets and two prognostic datasets. Based on the benchmark results, it is found that ChebyNet obtains the best overall performance on these six fault diagnosis datasets, and ChebyNet with EdegPool can also achieve the best results on the two prognosis datasets. Moreover, the performance of all models can be improved by using the frequency domain input. Finally, in the discussion part, four issues related to GNNs are also discussed.

The main contributions of this work are summarized as follows:

1) **Practical guideline.** We provide a practical guideline on leveraging GNNs for realizing fault diagnostics and prognostics. In this guideline, two common tasks on graphs are incorporated. Seven GNNs and four graph pooling methods are detailed. Besides, three types of graph construction methods for converting time series into graph data are also provided.
2) **GNN based framework.** We provide a novel intelligent fault diagnostics and prognostics framework based on GNNs. The framework consists of two branches, that is, the node-level fault diagnostics architecture and graph-level fault diagnostics or regression architecture. In node-level fault diagnosis, each node of a graph is considered as a sample, while the entire graph is considered as a sample in graph-level fault diagnosis.
3) **Benchmark study.** We evaluate the aforementioned GNNs on eight datasets and provide the benchmark results, where the eight datasets are composed of six fault diagnosis datasets and two prognosis datasets. Based on the benchmark results, four issues related to GNNs are also discussed.
4) **Open source code and datasets.** We provide a reproducible GNN-based benchmark code library, which allows researchers to easily add new models to arbitrary datasets. Besides, to validate and apply GNNs for fault diagnosis, two new fault diagnosis datasets, namely XJTUSpurgear and XJTUGearbox, have also been released.

The remainder of this paper is organized as follows. The theoretical background of GNNs and graph pooling methods are reviewed in Section 2. Then, a novel intelligent fault diagnostics and prognostics framework based on GNN is proposed in Section 3. Moreover, the benchmark study of the reviewed GNNs is carried out in Section 4. After that, four issues related to GNNs are discussed in Section 5, and some further directions are provided in Section 6. Finally, the conclusions of this paper are drawn in Section 7.

2. Graph neural networks

In fault diagnostics and prognostics, most conventional DNN-based methods focus on extracting temporal features from collected signals, while the spatial features between samples or multi-sensors are not taken seriously [50]. To model the spatial features, many works try to use CNNs to capture the spatial features [51–53], however, owing to its inherent working mechanism, it is not feasible for CNNs to embed the latent relationships explicitly between samples or multi-sensors into the obtained spatial features. Compared with these methods, GNNs have the ability to propagate the node information through the edges of a graph and learn a promising node or graph representations, where the edges of the graph reflect the relationships between samples or multi-sensors. For example, as shown in Fig. 2, during a general convolution operation, the convolution kernel slides on the signal without considering the relationship between each channel. While in the process of graph convolution, it aggregates the information of connected neighbor nodes along the weighted edges. Therefore, it is meaningful to discuss such an emerging DL technique in fault diagnostics and prognostics. In this section, we will outline the basic concepts of graph and review some commonly used GNNs and graph pooling methods.

2.1. Background

The concept of GNN was firstly proposed by Scarselli et al. [54] in 2009, which was defined in a recurrent architecture. Later, it was further elaborated by Gallicchio et al. [55] in 2010. The working mechanism of these earlier GNNs is based on the fixed point theory, which makes the training process of GNNs computationally expensive. Lately, more and more efforts have been made to overcome these challenges [56,57].

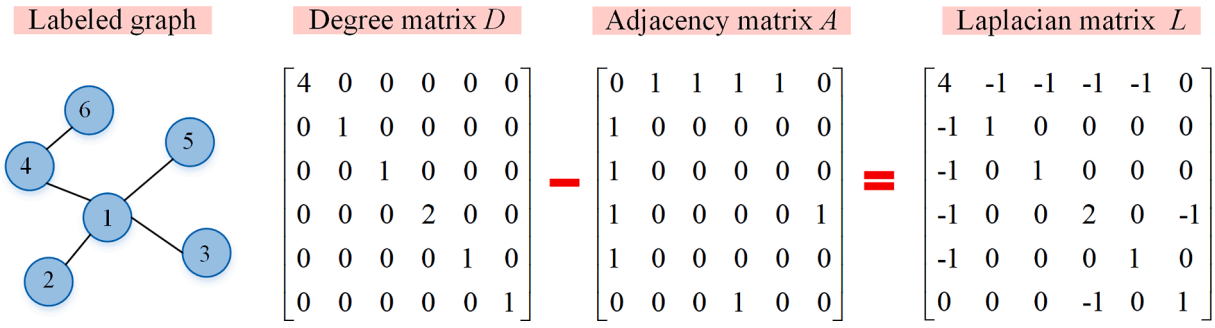


Fig. 3. The calculation of Laplacian matrix.

Table 1
The commonly used notations.

Notations	Descriptions
$\|\|$	Concatenation operation.
$*_G$	Graph convolution operation.
$\odot$	Element-wise Hadamard product
$\cdot$	Matrix multiplication
G	A graph.
$\mathbf{X} \in \mathbb{R}^{n\times d}$	Feature matrix of a graph.
$\mathbf{x} \in \mathbb{R}^{n}$	The feature vector of a graph, while $d = 1$.
$\mathbf{x}_i \in \mathbb{R}^{d}$	The node feature of the i-th node.
E	Edge set.
$\mathbf{A} \in \mathbb{R}^{n\times n}$	Adjacency matrix of a graph.
n	Number of nodes.
d	Dimension of node features.
$A_{ij} \in E$	Edge feature of the edge $(\mathrm{v_i}, \mathrm{v_j})$.
$\mathbf{D}$	Degree matrix of a graph.$\mathrm{D_{ii}} = \sum_j \mathrm{A_{ij}}$
$\mathbf{L}$	Laplacian matrix of a graph. $L = D\text{-}A$
$\mathbf{U}$	Eigenvector matrix of L.
$\mathbf{\Lambda}$	Eigenvalue matrix of L, which consists of eigenvalue λ.
$\mathrm{h}_v^{(k-1)}$	Hidden feature vector of node v at (k-1)-th layer.
$\boldsymbol{h}_{\mathbb{N}(v)}^{(k-1)}$	Aggregated node features of node v's immediate neighborhood.
$u \in \mathbb{N}(v)$	Neighbors of node v.
k	Layer index.
$\mathrm{H} \in \mathbb{R}^{n\times b}$	Output node feature matrix.
σ	Nonlinear activation function and ReLU is the commonly used one.
$\Theta, \theta, \varsigma$	Learnable model parameters.

Due to the successful application of CNN in many fields, a plenty of GCNs were proposed to redefine the notation of convolution for graph data. These GCNs can be divided into spectral-based GCNs and spatial-based GCNs [58]. The first remarkable spectral-based GCN was proposed by Brnna et al. [59], which extended convolution for graph data based on graph spectral theory. Since then, a large number of works have emerged to improve the performance of spectral-based GCNs [60–62]. The study of spatial-based GCNs was much earlier than spectral-based GCNs, which defined convolution operations directly on the graph and aggregated node information from neighborhoods. As early as 2009, Micheli [63] first used a constructive neural network for learning in structured domains, and recently, many spatial-based GCNs have been developed [64–66]. Moreover, a more comprehensive taxonomy of GNNs can be found in [67].

2.2. Mathematical notation of graph

In graph signal processing [68], a graph $\mathrm{G} = \mathrm{G(X,\ A,\ E)}$, where $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ represents the node feature matrix, E denotes the set of edges, n denotes the number of nodes and d is the feature length. $\mathbf{A} \in \mathbb{R}^{n\times n}$ denotes the adjacency matrix and the value of $A_{ij} = (v_i, v_j) \in E$. For an undirected graph, A_{ij} represents an edge connecting node v_i and v_j, while for a directed graph, A_{ij} denotes an edge directed from v_i to v_j. In addition to the adjacency matrix $\boldsymbol{A}$, the graph can also be represented by the Laplacian matrix $\mathbf{L}$ and the degree matrix $\mathbf{D}$, which can be obtained by Eq. (1), and an illustration of the aforementioned three matrices is shown in Fig. 3. Besides, some notations used in this paper are illustrated in Table 1.

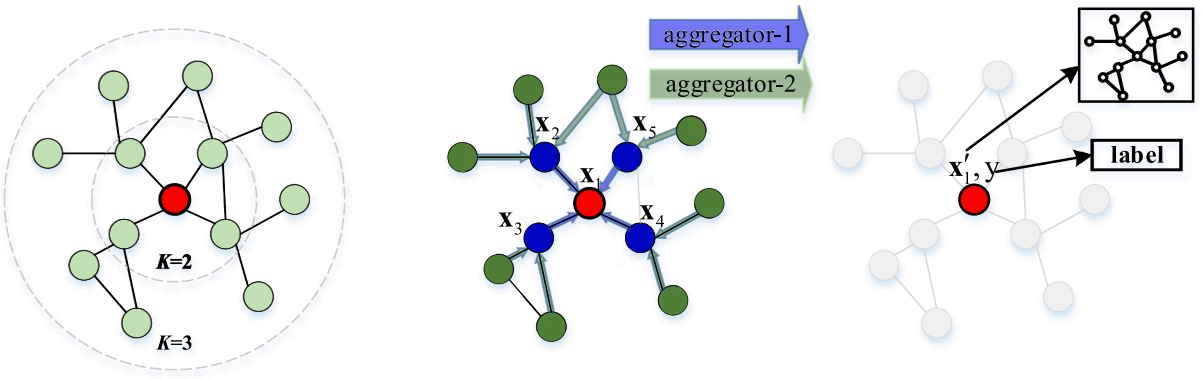


Fig. 4. The graph convolution process of ChebyNet.

$$\begin{cases} \boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \\ D_{ii} = \sum_j A_{ij} \end{cases} \tag{1}$$

2.3. Graph convolutional networks

In this paper, we focus on the most commonly used GNNs (i.e., GCNs), and three spectral-based GCNs and four spatial-based GCNs will be evaluated.

2.3.1. Spectral-based GCNs

The spectral based approaches use graph spectral filters to smooth the input signal of a node, which makes the features between adjacent nodes on the graph similar, thereby facilitating the subsequent tasks.

1) ChebyNet

The spectral graph convolution of a graph signal $x \in \mathbb{R}^n$ is defined as:

$$(\boldsymbol{x} *_G \boldsymbol{g})_\theta = \boldsymbol{U}((\boldsymbol{U}^T\boldsymbol{x}) \odot (\boldsymbol{U}^T\boldsymbol{g})) = \boldsymbol{U}\boldsymbol{g}_\theta\boldsymbol{U}^T\boldsymbol{x} \tag{2}$$

where $*_G$ represents the graph convolution operator, and $\boldsymbol{g}_\theta = \mathrm{diag}(\theta)$ is a filter parameterized by θ. $\odot$ is the element-wise Hadamard product. $\mathbf{U}$ is the eigenvector matrix of the normalized $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$. $\boldsymbol{I}_n$ is an identity matrix and n is the number of nodes.

However, the computational complexity of the filter $\boldsymbol{g}_\theta$ is large, and it is also not localized in space. Therefore, Defferrard et al. [61] proposed to use Chebyshev polynomials to approximate the filter and derived graph convolution of ChebyNet, which can be denoted as:

$$\boldsymbol{g}_\theta = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\boldsymbol{\Lambda}}) \tag{3}$$

$$\boldsymbol{h} = \boldsymbol{U}\boldsymbol{g}_\theta\boldsymbol{U}^T\boldsymbol{x} = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\boldsymbol{L}})\boldsymbol{x} \tag{4}$$

where K represents the order of the Chebyshev polynomials. $\boldsymbol{\Lambda} = \mathrm{dig}([\lambda_0, \lambda_1, ..., \lambda_{n-1}])$ and λ_{n-1} represent the eigenvalue matrix and eigenvalues of $\mathbf{L}$, respectively. $\widetilde{\boldsymbol{\Lambda}} = 2\boldsymbol{\Lambda}/\lambda_{max} - \boldsymbol{I}_n$ and $\widetilde{\boldsymbol{L}} = 2\boldsymbol{L}/\lambda_{max} - \boldsymbol{I}_n$ are the rescale eigenvalue matrix and Laplacian matrix, respectively. $\mathrm{T_k}$ denotes the Chebyshev polynomials, and $T_k\left(\widetilde{L}\right) = \mathrm{U}T\left(\widetilde{\Lambda}\right)\mathrm{U}^T$.

The graph convolution process of ChebyNet is depicted in Fig. 4. As can be seen from this figure, in the process of graph convolution, ChebyNet first determines the range of the aggregation neighborhood, that is, K takes 2. Then, the new node representation can be obtained by iteratively aggregating the information of its neighbors. Finally, the node classification task can be realized by using the learned node representations.

2) GCN

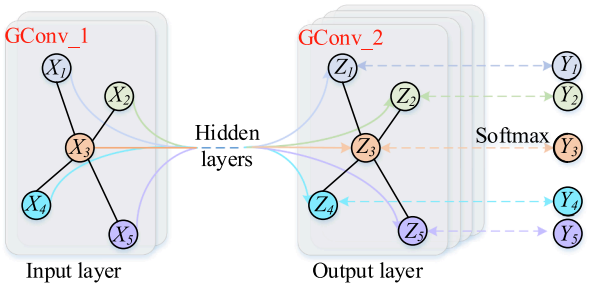


Fig. 5. The standard architecture of two-layer GCN.

Sharing the same idea with ChebyNet, Kipf et al. [62] also proposed to use Chebyshev polynomials to approximate the filter. But in addition, they further simplified the Chebyshev polynomials by making $\lambda_{max} \approx 2$, and under this approximation Eq. (4) becomes to:

$$\boldsymbol{h} = \theta_0 \boldsymbol{x} + \theta_1 (\boldsymbol{L} - \boldsymbol{I}_n)\boldsymbol{x} = \theta_0 \boldsymbol{x} - \theta_1 \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\boldsymbol{x} \tag{5}$$

At the same time, let $\theta = \theta_0 = -\theta_1$ and the above equation becomes to:

$$\boldsymbol{h} = \theta(\boldsymbol{I}_n + \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\boldsymbol{x} \tag{6}$$

To alleviate the problem of exploding/vanishing gradients, they further reduced $\boldsymbol{I}_n + \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ to $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, and then conducted the final expression of the commonly used graph convolutional network (GCN), which is mathematically defined as:

$$\boldsymbol{H} = \sigma\left(\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\boldsymbol{X}\boldsymbol{\Theta}\right) \tag{7}$$

where $\boldsymbol{\Theta}$ denotes the learnable parameters, and $\mathbf{H}$ is the convolved signal matrix. σ denotes the nonlinear activation function, and a standard two-layer GCN is shown in Fig. 5.

3) SGCN

The standard graph convolutional (GConv) layer is defined by Eq. (7). A two-layer GCN, as shown in Fig. 5, can be obtained by stacking two GConv layers and its definitions are:

$$\boldsymbol{Y}_{\mathrm{GCN}} = \mathrm{softmax}(\boldsymbol{\Gamma}(\sigma(\boldsymbol{\Gamma}\boldsymbol{H}^0\boldsymbol{\Theta}^0))\boldsymbol{\Theta}^1), \qquad \boldsymbol{\Gamma} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{8}$$

where $\boldsymbol{\Theta}^0$ and $\boldsymbol{\Theta}^1$ denote the learnable parameters of the first and second GConv layer, respectively. $\boldsymbol{H}^0 = \boldsymbol{X}$ and $\boldsymbol{Y}_{\mathrm{GCN}}$ denotes the predicted results.

In simplifying graph convolutional networks (SGCN), it turns out that the nonlinearity between GCN layers is not critical, but most of the benefits come from local averaging [69]. Therefore, in SGCN, the nonlinear activation function is removed and the weights are reparametrized as a single matrix, i.e., $\boldsymbol{\Theta} = \boldsymbol{\Theta}^0\boldsymbol{\Theta}^1 \ldots \boldsymbol{\Theta}^k$. The resulting two-layer SGCN becomes:

$$\boldsymbol{Y}_{\mathrm{SGCN}} = \mathrm{softmax}(\boldsymbol{\Gamma}^2\boldsymbol{X}\boldsymbol{\Theta}) \tag{9}$$

where $\boldsymbol{Y}_{\mathrm{SGCN}}$ denotes the predicted results of SGCN.

2.3.2. Spatial-based GCNs

The goal of the spatial-based approaches is to define convolution operations directly on the graph and aggregate node information from neighborhoods, which forms the high-level representation of a node rather than defining the convolution operation in the Fourier domain. This makes the spatial-based approaches faster than the spectral-based methods and can be easily generalized to other graphs.

1) GraphSage

In GraphSage (Sample and aggregate) [64], two critical functions are defined, that is, the AGGREGATE (·) function and CONCAT (·) function. The AGGREGATE (·) function is used to aggregate information from node neighbors, and CONCAT (·) function is utilized to concatenate the current node features with the aggregated node features.

For example, for a node v with node neighbors $\forall u \in \mathbb{N}(v)$, its node features after k iterations of aggregation become:

$$\boldsymbol{h}_{\mathbb{N}(v)}^{(k)} = \mathrm{AGGREGATE}_k(\{\boldsymbol{h}_u^{(k-1)}, \forall u \in \mathbb{N}(v)\}) \tag{10}$$

$$\boldsymbol{h}_v^{(k)} = \sigma(\boldsymbol{\Theta}_k \cdot \mathrm{CONCAT}_k(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{h}_{\mathbb{N}(v)}^{(k)})) \tag{11}$$

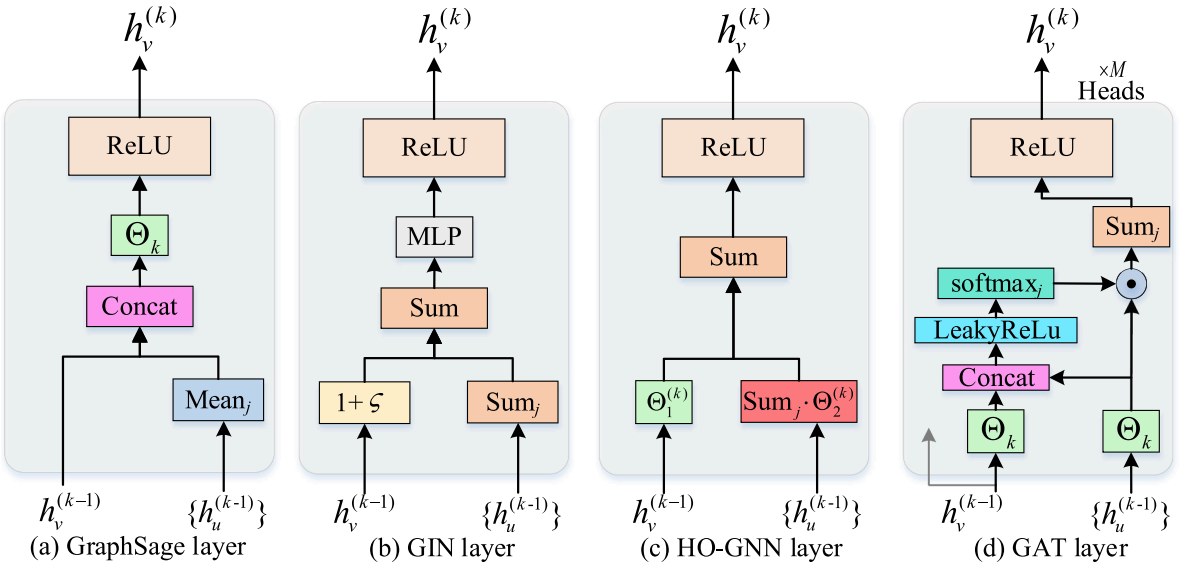


Fig. 6. The feature update mechanism of four different spatial-based GCNs.

where $\boldsymbol{h}_{\mathbb{N}(v)}^{(k)}$ and $\boldsymbol{h}_{u}^{(k-1)}$ represent the aggregated node features of node v's adjacent neighborhood and the node features of a node v's adjacent node, respectively. $\boldsymbol{h}_{v}^{(k)}$ denotes the node features of node v after k iterations of aggregation, where $\boldsymbol{h}_{v}^{(0)} = \boldsymbol{x}_v$. In addition, the mean aggregator, max aggregator and sum aggregator are three widely used AGGREGATE (·) functions, and in this paper the mean aggregator is adopted in GraphSage.

The detailed process of GraphSage layer is depicted in Fig. 6(a). In GraphSage layer, it first uses mean aggregator to acquire the mean value of the features of the neighboring nodes of node v. After that, it concatenates the raw features of node v with the features obtained by the mean aggregator to obtain a fused feature. Finally, the new features of node v can be obtained by multiplying the fused feature with a learnable parameter $\boldsymbol{\Theta}_k$ and using ReLU for nonlinear activation.

2) GIN

In Graph Isomorphism Network (GIN) [66], it was found the GCN variants (e.g., GraphSage) fail to learn the discriminative features of two isomorphic graphs, and using sum aggregator can learn a more precise structural information of graphs. Therefore, they proposed to use sum aggregator as the AGGREGATE (·) function, and afterwards use the MLP (Multilayer perceptron) to model the CONCAT (·) function. Then, the resulting graph isomorphism convolutional (GINConv) layer for learning node representations is mathematically defined as:

$$\boldsymbol{h}_{v}^{(k)} = \sigma\left(\mathrm{MLP}^{(k)}\left((1+\varsigma^{(k)})\boldsymbol{h}_{v}^{(k-1)} + \sum_{\mathbb{N}(v)} \boldsymbol{h}_{u}^{(k-1)}\right)\right) \tag{12}$$

where ς represents a learnable parameter.

The detailed process of GINConv layer is shown in Fig. 6(b). In GIN layer, the raw feature of node v is first multiplied by a learned parameter ς, and then a sum aggregator is used to aggregate its neighbor nodes' features. After that, these two features are added and inputted into an MLP layer. Finally, the output result of MLP is activated nonlinearly and used as the new feature of node v.

3) HO-GNN

The standard GCN and GraphSage only learn node representations at a single granularity. To admit the model to capture graph structures of varying granularity, a higher-order graph neural network (HO-GNN) with the goal to learn features of different orders from k-GNNs was proposed in [70], which can be defined as:

$$\boldsymbol{h}_{o,L}^{(k)}(s) = \sigma\left(\boldsymbol{h}_{o,L}^{(k-1)}(s)\cdot\boldsymbol{\Theta}_1^{(k)} + \sum_{u\in\mathbb{N}_L(s)} \boldsymbol{h}_{o,L}^{(k-1)}(u)\cdot\boldsymbol{\Theta}_2^{(k)}\right) \tag{13}$$

where $\boldsymbol{h}_{o,L}^{(k)}(s)$ is the learned node representations of HO-GNN, and $o\geqslant 2$ represents the dimension of the subgraph at the k-th iteration. s represents a subgraph composed of o nodes and u denotes the neighbor subgraph of graph s. $\mathbb{N}_L(s)$ denotes the local neighborhood of

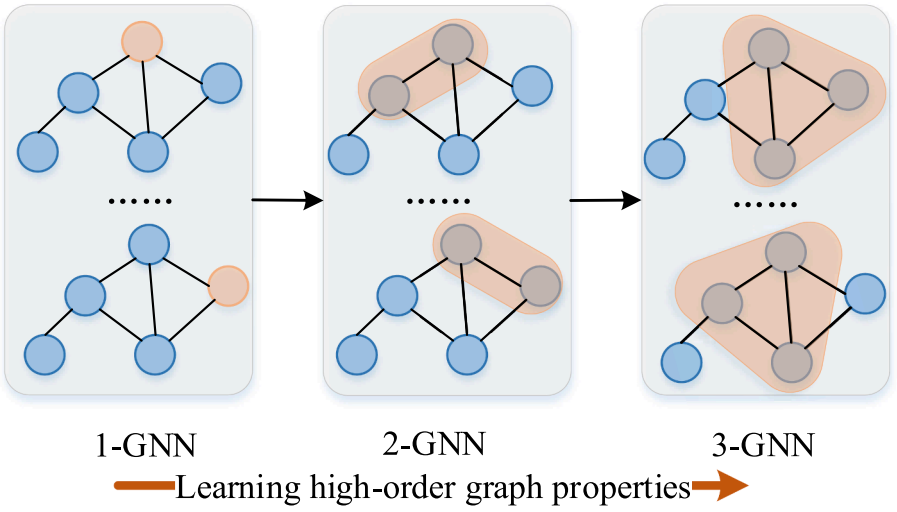


Fig. 7. An illustration of HO-GNN. In 1-GNN, each node is considered as a node; in 2-GNN, two nodes are considered as one node; in 3-GNN, three nodes are considered as one node.

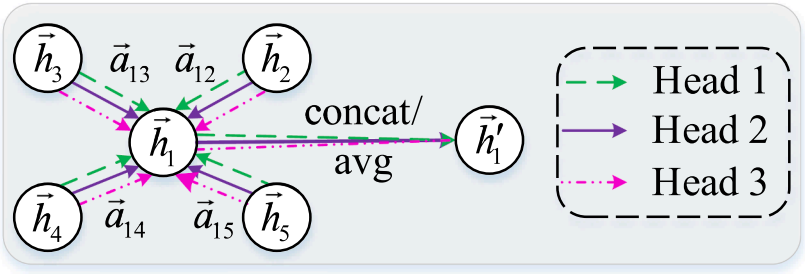


Fig. 8. An illustration of multi-head graph attention (the number of heads $M = 3$).

subgraph s.

The feature update mechanism of HO-GNN is shown in Fig. 6(c), and the detailed process of HO-GNN layer is shown in Fig. 7. As can be seen in Fig. 7, in each order, a feature $\boldsymbol{h}_{o,\mathcal{L}}(s)$ is learned for each subgraph s on o nodes, and the feature $\boldsymbol{h}_{o,\mathcal{L}}(s)$ is initialized with the learned features of all (o-1)-element subgraphs of s. Through such a process, a hierarchical representation of the input graph can be learned.

4) GAT

Graph attention network (GAT) [65] introduced attention mechanisms into graph convolution operation. For a graph with a set of node features $\boldsymbol{h} = \{\vec{\boldsymbol{h}}_1, \vec{\boldsymbol{h}}_2, ..., \vec{\boldsymbol{h}}_n\}$, $\vec{\boldsymbol{h}}_i \in \mathbb{R}^d$. The output of graph attention convolutional (GATConv) layer is a new set of node features, i.e., $\boldsymbol{h}' = \{\vec{\boldsymbol{h}'}_1, \vec{\boldsymbol{h}'}_2, ..., \vec{\boldsymbol{h}'}_n\}$, $\vec{\boldsymbol{h}'}_i \in \mathbb{R}^{d'}$, where d' is the length of new node features. Therefore, the attention coefficients of each node can be expressed as:

$$\alpha_{ij} = \frac{exp(LeakReLU(\vec{\boldsymbol{a}}^T \cdot [\boldsymbol{\Theta}\vec{\boldsymbol{h}}_i || \boldsymbol{\Theta}\vec{\boldsymbol{h}}_j]))}{\sum_{u \in \mathbb{N}_i} exp(LeakReLU(\vec{\boldsymbol{a}}^T \cdot [\boldsymbol{\Theta}\vec{\boldsymbol{h}}_i || \boldsymbol{\Theta}\vec{\boldsymbol{h}}_u]))} \tag{14}$$

where $\vec{\boldsymbol{a}}^T$ denotes the parameter of a single-layer feedforward neural network, $u \in \mathbb{N}_i$ means the neighbors of node i, and α_{ij} denotes the attention coefficient between node i and node j. $||$ denotes the concatenation operation and $LeakReLU(\cdot)$ is a nonlinear activation function.

Based on the calculated attention coefficients, a multi-head attention graph convolutional layer is defined as:

$$\vec{\boldsymbol{h}'}_i = \Big\|_{m=1}^{M} \sigma\left(\sum_{j \in \mathbb{N}_i} \alpha_{ij}^m \boldsymbol{\Theta}^m \vec{\boldsymbol{h}}_j\right) \tag{15}$$

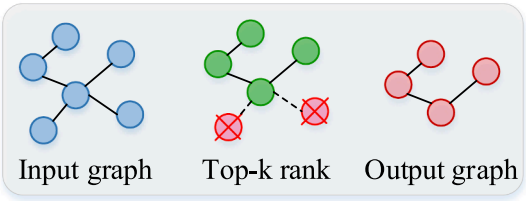


Fig. 9. An illustration of TopkPool.

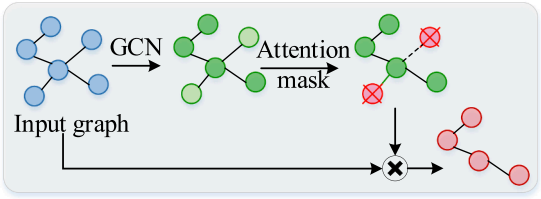


Fig. 10. An illustration of SAGPool.

where M means the number of multi-head attention, and α_{ij}^{m} is the normalized attention coefficients computed by the m-th attention mechanism. $\overrightarrow{h}_i$ represents the finally learned node representations.

The detailed process of GAT layer is shown in Fig. 6(d). In GAT layer, it first inputs the raw feature of node v and the features of its neighborhood into a softmax layer to obtain the attention coefficients of each node. Then, the new node features node v can be obtained by the inner product of the generated attention coefficient and the features of its corresponding adjacent nodes. If multi-head attention is adopted, the new feature of node v is the concatenation or average of node features obtained under different attentions, and an illustration of multi-head graph attention is depicted in Fig. 8.

2.4. Graph pooling layer

Graph pooling is desirable in graph classification task or regression, which can achieve dimensionality reduction and hierarchical learning. With the assistance of a graph pooling layer, the number of nodes in a graph is reduced and hence the computation complexity is reduced. Besides, some graph pooling layers are able to enhance the hierarchical representation of data by establishing a model that can learn node assignment from its features or topology in each layer, and in this paper, four graph pooling methods are discussed.

1) TopkPool

To achieve graph pooling, TopkPool [71] defines a projection score $\overrightarrow{p}$ at first, then the node features are projected on the vector $\overrightarrow{p}$ as the importance of nodes. Finally the top k nodes with the highest scores are retained, as shown in Fig. 9. Therefore, the pooled graph $(\boldsymbol{X}^{'},\boldsymbol{A}^{'})$ can be expressed as:

$$\begin{cases} \boldsymbol{X}^{'} = (\boldsymbol{X} \odot tanh(\boldsymbol{z}))_{i,:} \\ \boldsymbol{A}^{'} = \boldsymbol{A}_{i,i} \\ i = \text{top} - k(\boldsymbol{z}, k), \boldsymbol{z} = \boldsymbol{X} \cdot \overrightarrow{\boldsymbol{p}} / \|\overrightarrow{\boldsymbol{p}}\|^2 \end{cases} \tag{16}$$

where $k \in (0, 1]$ denotes the pooling ratio, $\mathbf{z}$ represents the importance matrix, and $i, :$ is an indexing operation which takes slices at indices specified by i. $\|\cdot\|^2$ represents the L_2 norm. top $-k(\cdot)$ denotes the top k rank mechanism.

2) SAGPool

Different from TopkPool that uses projected scores to evaluate the importance of nodes, self-attention pool (SAGPool), as illustrated in Fig. 10, uses graph convolution to obtain the importance of nodes, which not only considers the features of the node itself, but also considers the influence of the structural information of the graph [72]. Therefore, the pooled graph $(\boldsymbol{X}^{'},\boldsymbol{A}^{'})$ can be defined as:

$$\begin{cases} \boldsymbol{Z} = \sigma(\text{GConv}_{\boldsymbol{\Theta}}(\boldsymbol{X}, \boldsymbol{A})) \\ idx = \text{top} - k(\boldsymbol{Z}, \lceil kn \rceil) \\ \boldsymbol{Z}_{mask} = \boldsymbol{Z}_{idx} \end{cases} \tag{17}$$

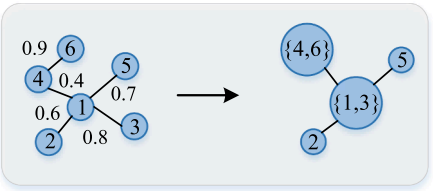


Fig. 11. An illustration of EdgePool.

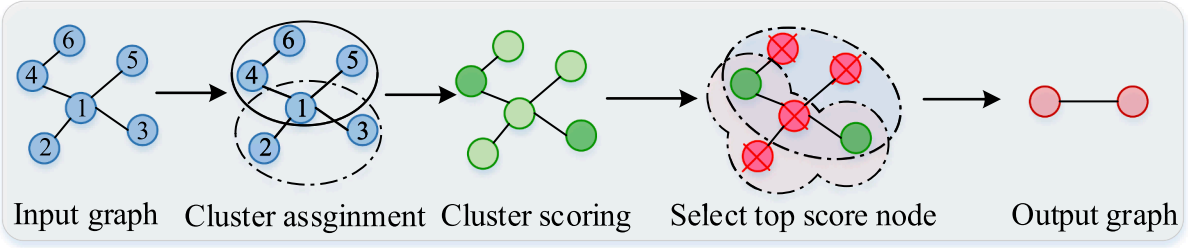


Fig. 12. An illustration of ASAPool.

where $\mathbf{Z}$ denotes the attention score obtained by the graph convolution (GConv) and $\boldsymbol{\Theta}$ denotes the learnable parameters. top$-k(\cdot)$ returns the indices of $\lceil kn \rceil, k \in (0,1]$, *idx* denotes an indexing operation and $\boldsymbol{Z}_{mask}$ represents the obtained attention mask.

$$\begin{cases} \widetilde{\boldsymbol{X}} = \boldsymbol{X}_{idx,:} \boldsymbol{X}' = \widetilde{\boldsymbol{X}} \odot \mathbf{Z}_{mask} \boldsymbol{A}' = \boldsymbol{A}_{idx,idx} \end{cases} \tag{18}$$

where $\boldsymbol{X}_{idx,:}$ is the node-wise indexed feature matrix.

3) EdgePool

In EdgePool [73], it iteratively merges the nodes on each edge pair by pair to form a new node, and retains the connection relationship of the merged nodes to the new node. The procedure of realizing EdgePool is shown in Fig. 11, which can be defined as follows.

$$\begin{cases} r(e_{ij}) = \boldsymbol{\Theta} \cdot (\boldsymbol{x}_i || \boldsymbol{x}_j) + \boldsymbol{b} \\ s_{ij} = 0.5 + \mathrm{softmax}_j(r_{ij}) \end{cases} \tag{19}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ denotes the node features of the i-th and j-th node, and $r(e_{ij})$ denotes the raw score of edge e_{ij}.$\boldsymbol{\Theta}$ and $\boldsymbol{b}$ are the learnable parameters. s_{ij} denotes the final edge score which is normalized over all edges of a node.

Then, the node features of the new node can be defined as:

$$\boldsymbol{x}_{ij} = s_{ij}(\boldsymbol{x}_i + \boldsymbol{x}_j) \tag{20}$$

Finally, by iteratively doing the process defined in the above two functions, the pooled graph can be obtained.

4) ASAPool

In order to capture the graph sub-structure or scale to large graphs during the process of graph pooling, adaptive structure aware pooling (ASAPool) [74] proposed to utilize a self-attention based convolution to capture the importance of each node in a given graph. As shown in Fig. 12, the pooled graph can be obtained by the following steps:

First, consider each node v_i in the graph as the center of a cluster $c_h(v_i)$, then learn a representation from each cluster by the designed self-attention mechanism, and the learned cluster representation can be defined as:

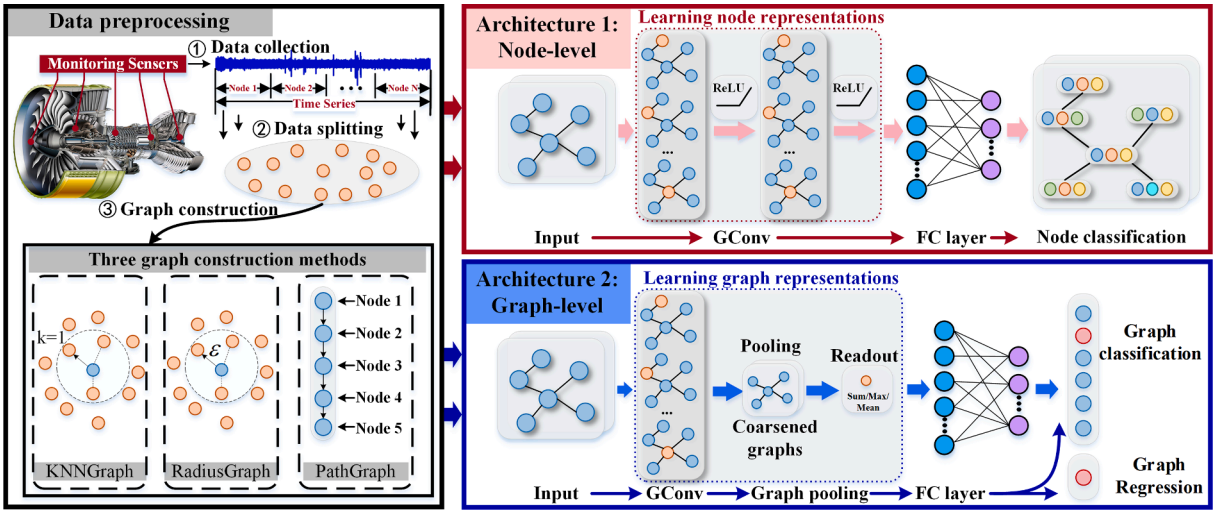


Fig. 13. The proposed intelligent fault diagnostics and prognostics framework based on GNN.

$$\begin{cases} m_i = f_m(\boldsymbol{x}'_j | v_j \in c_h(v_i)) \\ \alpha_{ij} = \mathrm{softmax}(\overrightarrow{\boldsymbol{\theta}}^T \sigma(\boldsymbol{\Theta} m_i || \boldsymbol{x}'_j)) \\ \boldsymbol{x}_i^c = \sum_{j=1}^{|c_h(v_i)|} \alpha_{ij} \boldsymbol{x}_j \end{cases} \tag{21}$$

where $\boldsymbol{x}'_j$ is the obtained structure information of a cluster $c_h(v_i)$, f_m is a master function, and m_i denotes a master query. $\overrightarrow{\boldsymbol{\theta}}^T$ and $\boldsymbol{\Theta}$ are learnable vector and matrix, respectively. α_{ij} is the calculated attention scores, and $\boldsymbol{x}_i^c$ is the learned cluster representation.

$$\begin{cases} \phi_i = \sigma(\boldsymbol{x}_i^c \boldsymbol{\Theta}_1 + \sum_{j \in \mathbb{N}(i)} (\boldsymbol{x}_i^c \boldsymbol{\Theta}_2 - \boldsymbol{x}_j^c \boldsymbol{\Theta}_3)) \\ \widetilde{\boldsymbol{X}}^c = \boldsymbol{\Phi} \odot \boldsymbol{X}^c \widetilde{i} = \mathrm{top} - k(\widetilde{\boldsymbol{X}}^c, \lceil kn \rceil) \quad \boldsymbol{X}' = \widetilde{\boldsymbol{X}}^c_{i,:} \quad \boldsymbol{A}' = \widetilde{\boldsymbol{S}}^T \widetilde{\boldsymbol{A}}^c \widetilde{\boldsymbol{S}}, \quad \widetilde{\boldsymbol{S}} = \boldsymbol{S}_{:,\widetilde{i}} \end{cases} \tag{22}$$

where $\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \boldsymbol{\Theta}_3$ are learnable parameters, ϕ_i denotes the fitness score and $\boldsymbol{\Phi} = [\phi_1, \phi_2, ..., \phi_N]^T$ is the fitness vector. $\widetilde{i}$ denotes the indices of top $\lceil kn \rceil$ selected clusters, $\boldsymbol{S}$ and $\widetilde{\boldsymbol{S}}$ denote the raw and pruned cluster assignment matrix, respectively.

3. Intelligent fault diagnostics and prognostics framework based on GNN

The proposed intelligent fault diagnostics and prognostics framework based on GNN is shown in Fig. 13, which builds upon PyTorch [75] and PyTorch Geometrics [76]. This framework can be divided into two parts, that is, data preprocessing, and fault diagnostics and prognostics based on GNNs. In data preprocessing, the three types of graph construction methods (i.e., KNNGraph, RadiusGraph and PathGraph) are provided for constructing graphs from time series. Two tasks on graphs, that is, node classification and graph classification or regression, are incorporated with fault diagnostics and prognostics, and the corresponding architectures are designed. Moreover, the aforementioned seven GNNs and four pooling methods are also integrated into this framework. Finally, a benchmark study is also carried out on eight datasets, including six fault diagnosis datasets and two prognosis datasets.

In detail, to realize intelligent fault diagnostics and prognostics, the collected raw data is firstly divided into subsamples, then a graph construction method is utilized to transform the subsamples into graphs. After that, fault diagnostics and prognostics can be realized with the proposed architectures. For fault diagnosis, if the input data is univariate data, it is recommended to use the architecture 1 proposed in the framework and each node of a graph is considered as a sample at this condition. While, if the input data is multivariate data, it is recommended to use the architecture 2 and the whole graph is considered as a sample at this moment. But for prognostics, only the architecture 2 can be adopted to obtain the graph representations for regression.

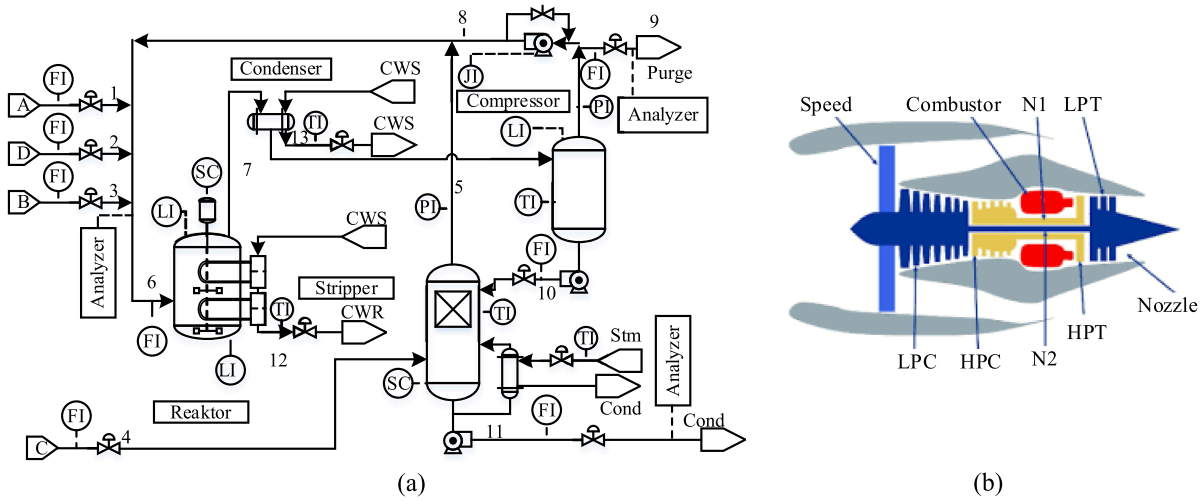


Fig. 14. Multiple sensors in typical industrial processes or systems.(a) Tennessee Eastman process [77];(b) Turbofan engine system [78].

3.1. Constructing graphs from time series

In practice, two kinds of time series are often encountered, namely univariate time series and multivariate time series. For example, due to space structure limitations, only a few sensors can be mounted on rotating machinery, which makes the collected high-frequency vibration signals usually univariate time series. Hence, in vibration signal-based fault diagnosis or prognosis scene, high-frequency vibration signals are divided into subsamples to construct a graph. While for industrial processes or systems, multiple sensors are usually used to collect low-frequency signals of multiple source physical quantities [79], as shown in Fig. 14. In the process of graph construction, the univariate time series will be divided into subsamples, and then transform these samples into graphs by calculating the distance between each sample. In contrast, when using multivariate time series to construct graph dataset, each sensor is directly treated as a node and the edge between each node is determined based on the calculated similarity of multiple sensors.

1) Constructing graphs from univariate time series

To construct graphs from univariate time series, the raw data with length l is firstly divided into subsamples with length d, and each subsample is assigned with the corresponding label. It is worth noting that there is no overlap between each sample, and the obtained subsample set can be denoted as:

$$\Pi = [(\boldsymbol{x}_1, \mathrm{y}_1), (\boldsymbol{x}_2, \mathrm{y}_2), ..., (\boldsymbol{x}_n, \mathrm{y}_n)], \quad n = \lceil \frac{l}{d} \rceil \tag{23}$$

where Π is the obtained sample set, $\mathbf{x}$ denotes the subsample and y denotes the label. n represents the number of subsamples.

After obtaining the subsample set, a graph with τ nodes can be constructed by considering each subsample as a node in the graph. This means that we first need to select τ samples, and then find the neighbors of each sample. In this paper, three methods are provided to find the neighbors of each subsample, that is, the K-nearest neighbor graph (KNNGraph), RadiusGraph, and PathGraph.

KNNGraph. In KNNGraph, the top-k nearest neighbors are found for each node, and the obtained neighbors of node $\boldsymbol{x}_i$ can be denoted as:

$$\mathrm{Ne}(\boldsymbol{x}_i) = \mathrm{KNN}(k^{'}, \boldsymbol{x}_i, \Psi) \tag{24}$$

where KNN $(\cdot)$ returns the top-k nearest neighbors of node $\boldsymbol{x}_i$ in set Ψ, and $k^{'}$ takes 5 in this paper.$\Psi = [\boldsymbol{x}_{i+1}, \boldsymbol{x}_{i+2}, ..., \boldsymbol{x}_{i+m}]$ denotes a subset with m samples, and $\mathrm{Ne}(\boldsymbol{x}_i)$ represents the neighbors of node $\boldsymbol{x}_i$.

The edge weight between each node of the KNNGraph can be estimated by a Gaussian kernel weight function, which is defined as:

$$e_{ij} = exp\left(- \frac{||(\boldsymbol{x}_i, \boldsymbol{x}_j)||^2}{2\zeta^2} \right), \quad \boldsymbol{x}_j \in \mathrm{Ne}(\boldsymbol{x}_i) \tag{25}$$

where e_{ij} denotes the edge weight between node $\boldsymbol{x}_i$ and node $\boldsymbol{x}_j$. ζ is the bandwidth of the Gaussian kernel.

RadiusGraph. In RadiusGraph, the cosine similarity is utilized to estimate the distance between samples, and a threshold ϵ is defined. If the cosine similarity is greater than the threshold, there will be an edge between the two nodes. Therefore, the neighbors of node $\boldsymbol{x}_i$ can be obtained by:

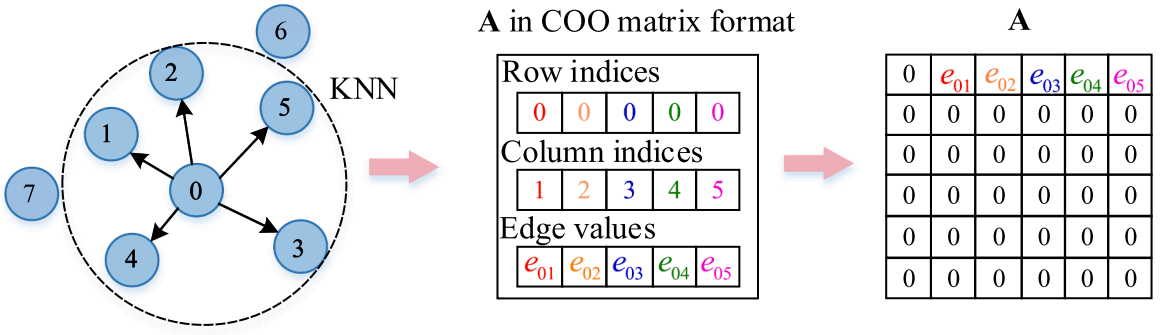


Fig. 15. Example of adjacency matrix construction based on KNNGraph.

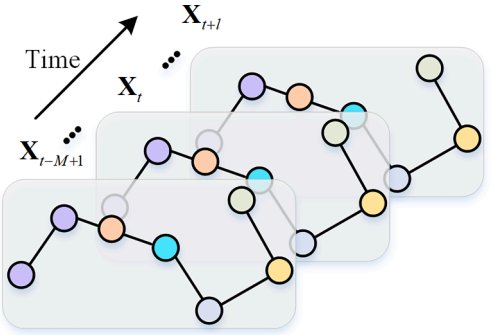


Fig. 16. An illustration of generating spatial–temporal graphs on the constructed sensor network.

$$\mathrm{Ne}(\boldsymbol{x}_i) = \epsilon \ominus -\mathrm{radius}(\mathbf{x}_i, \Psi), \quad \text{if} \quad \epsilon \ominus -\mathrm{radius}(\mathbf{x}_i, \Psi) > \epsilon \tag{26}$$

where $\epsilon \ominus -\mathrm{radius}(\cdot)$ calculates the cosine similarity between node $\boldsymbol{x}_i$ and node in set Ψ, and returns the neighbors of node $\boldsymbol{x}_i$. The value of threshold ϵ takes 0 here.

PathGraph. In PathGraph, the samples in the sub-set Ψ are connected sequentially, and there is an edge between every two adjacent samples. Besides, the edge weight can also be estimated by Eq. (25).

By using the above equations, we can find the neighbors of each subsample, thereby forming the corresponding adjacency matrix denoted by coordinate (COO) format [80]. For example, through the calculation of KNNGraph, it can be obtained that the adjacent nodes of node 0 are node 1, 2, 3, 4, 5 and the corresponding edges values are $[e_{01}, e_{02}, e_{03}, e_{04}, e_{05}]$, as shown in Fig. 15. After that, we can derive the corresponding adjacency matrix **A** based on the obtained COO matrix. It is worth mentioning that the adjacency matrices of the constructed graphs are all expressed in COO format in our open source code.

2) Constructing graphs from multivariate time series

In real industrial scenarios, a aplenty of sensors could be used to monitor the state of a system, and these sensors can naturally form a sensor network. Besides, the sensor network continuously generates spatial–temporal graphs about the health state of the system. Therefore, the multivariate time series collected by different sensors can be constructed as a sensor network, and the spatial–temporal graphs can be generated on this network for model training.

For multivariate time series $\boldsymbol{X} = [\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_\kappa]$ composed of κ sensors, the signal in each sensor is firstly normalized. By considering each sensor as a node in the sensor network, the KNNGraph or RadiusGraph can be constructed based on Eq. (24) or Eq. (26). After obtaining the constructed sensor network, the spatial–temporal graphs can be generated by assigning time series collected at different time stamps as the node features, as demonstrated in Fig. 16.

3.2. The designed architecture

As mentioned before, the designed architectures for node-level task and graph-level task are different. In node-level task, each node in a graph is considered as a sample, and the model only needs to learn the representations of the nodes. However, in graph-level task, an entire graph is considered as a sample, and the model needs to learn node representation first, and then obtain the representation of the entire graph.

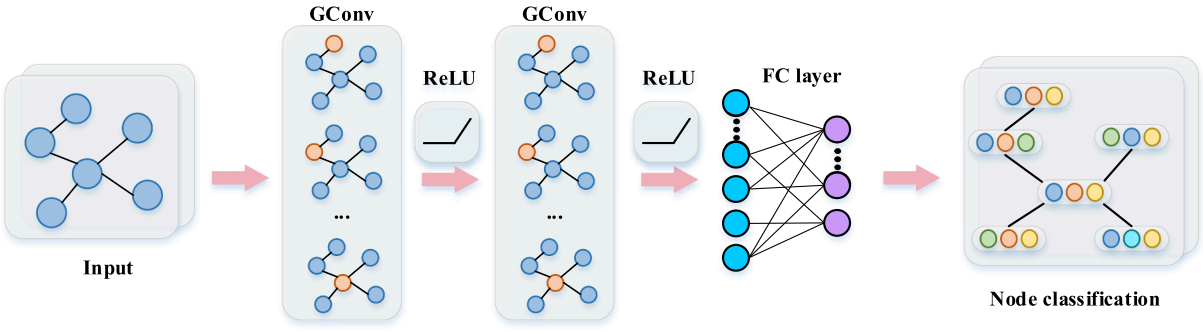


Fig. 17. The framework for node-level fault diagnostics.

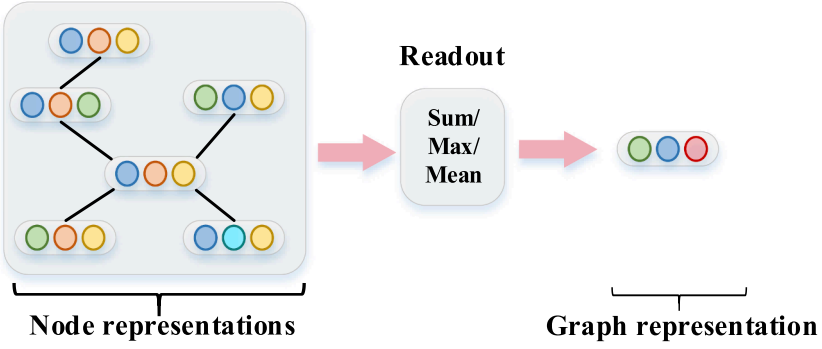


Fig. 18. An illustration of the readout layer. In the readout layer, the graph representation can be obtained by performing the sum/max/mean operation on the learned node representations.

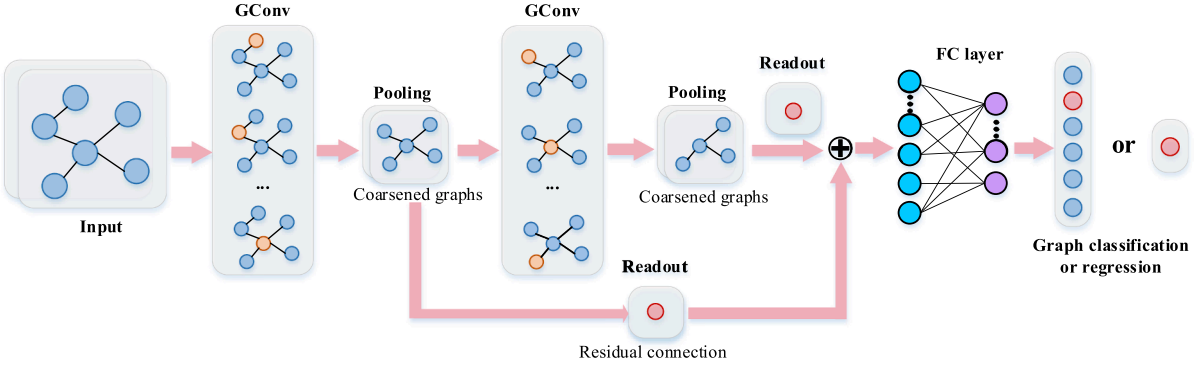


Fig. 19. The framework for graph-level fault diagnostics or prognostics.

1) The architecture for node-level fault diagnostics

The goal of node classification is to assign the correct target label to each node in a graph. To realize node-level fault diagnostics, GNNs are utilized to learn the feature representations of each node. In this paper, the designed architecture for node-level fault diagnostics is shown in Fig. 17, which consists of two graph convolutional (GConv) layers and two fully connected (FC) layers, where the GConv layers are utilized to learn the node representations and the FC layer is used for node classification.

The cross-entropy (CE) loss is usually utilized as the objective function. Therefore, the learning criterion of a multi-class node classification task is:

$$L_{CE}(\mathrm{y},\bar{y}) = -log\left(\frac{e^{y_{\bar{y}}}}{\sum_{j=1}^{C} e^{y_j}}\right) \tag{27}$$

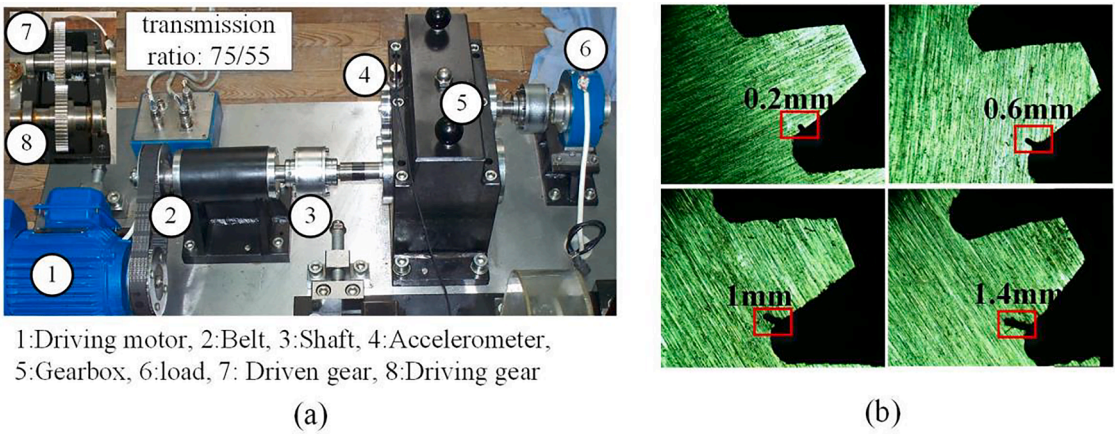


Fig. 20. The test rig of XJTUSpurgear Dataset. (a) The test rig; (b) Health conditions of spur-gears.

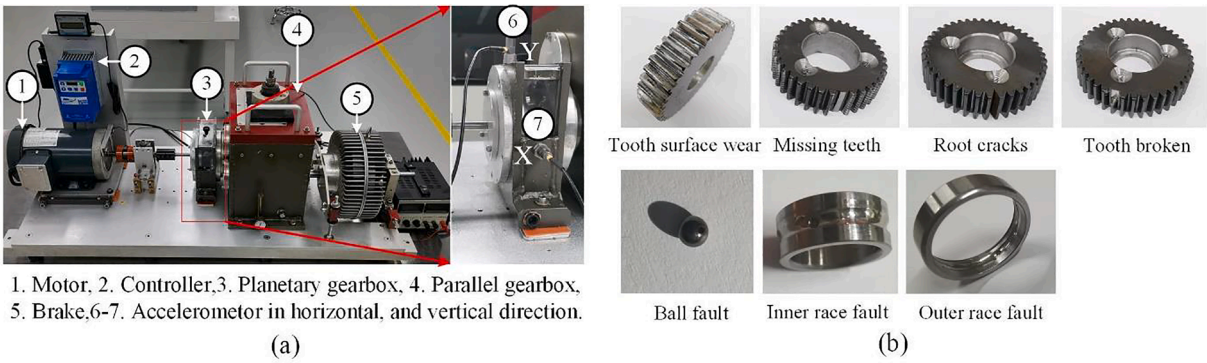


Fig. 21. The test rig of XJTUGearbox Dataset. (a) The test rig; (b) Health conditions of gears and bearings.

where C denotes the number of class, and y and $\overline{y} \in \{1, 2, \ldots, C\}$ represent the predicted results and the true class, respectively.

2) The architecture for graph-level fault diagnostics or prognostics

To achieve graph-level fault diagnostics or prognostics, GNNs with graph pooling layer and readout layer are utilized to obtain the representation of the entire graph, and a simple illustration of the readout layer is shown in Fig. 18. After obtaining the graph representation, the standard machine learning techniques can be performed on this representation.

The designed architecture for graph-level fault diagnostics or prognostics is shown in Fig. 19. As can be seen, the architecture consists of two GConv layers, two graph pooling layers, two readout layers and two FC layers. In this architecture, a GConv layer follows by a graph pooling layer to coarsen (downsample) the input graph into sub-graphs, which reduces the dimensionality of the input graph and speeds up the computation. After that, a readout layer collapses the node representations of the sub-graphs into a graph representation by using the sum/max/mean operation. Besides, the residual connection is also utilized to improve the performance of the model. Finally, the learned graph representation is inputted to the FC layer for realizing graph-level fault diagnostics or prognostics.

For multi-class graph classification, the objective function is the same with Eq. (27), and for graph regression, the Mean Square Error (MSE) is usually employed as the learning criterion:

$$L_{MSE}(\overline{\mathrm{A}}, \mathrm{A}) = \frac{1}{n_G} \sum_{i=1}^{n_G} (\overline{\mathrm{A}} - \mathrm{A})^2 \tag{28}$$

where n_G denotes the number of graphs, and $\overline{\mathrm{A}}$ and A represent the output and target vectors, respectively.

3.3. Datasets

This benchmark study is implemented on eight datasets, including two self-collected datasets, and six open source datasets.

Table 2
The descriptions of the six open source datasets.

Dataset	Research Object	Application	Description
CWRU [81]	Bearing	Fault diagnosis	The CWRU dataset consists of four sub-datasets with different working conditions, that is, 0 hp-1797 rpm, 1 hp-1772 rpm, 2 hp-1750 rpm, and 3 hp-1730 rpm. Within each working condition, single-point motor bearing faults, including ball fault, inner race fault, and outer race fault, are introduced. In our benchmark study, the first dataset is used which includes three types of ball fault, three types of inner race fault, and three types of outer race fault. Together with the normal state, it is considered as a 10-class classification task.
PU [82]	Bearing	Fault diagnosis	The data in the PU dataset can be divided into three types, that is, healthy bearings, artificially damaged bearings, and bearings that have been damaged through accelerated experiments. In our benchmark study, the data of one healthy bearing and twelve artificially damaged bearings are used, therefore, it can be considered as a 13-class classification task.
MFPT [83]	Bearing	Fault diagnosis	The MFPT dataset is composed of four sets of bearing vibration signals, including a baseline dataset, seven outer race fault datasets, seven inner race fault datasets, and some other datasets. In this study, the former three datasets are used, therefore, it can be considered as a 15-class classification task.
SEU [84]	Gearbox	Fault diagnosis	The SEU dataset consists of a gear dataset, and a bearing dataset, where the gear dataset contains six kinds of gear healthy states and the bearing dataset contains six kinds of bearing healthy states. Two different working conditions are included. In this study, each healthy state is considered as a fault type, so it becomes a 20-class classification task.
CMAPSS [85]	Turbofan engine	Prognosis	The CMAPSS dataset is an engine degradation simulation dataset. It consists of four different sub-sets, i.e., FD001, FD002, FD003, and FD004. 21 different sensor measurements are recorded. Each subset is composed of a training set, and a testing set, and the task of this dataset is to predict the real RUL of the testing set.
PHM2010 [86]	Milling cutters	Prognosis	The PHM2010 dataset is a dataset about tool wear during the operation of high-speed CNC machine tools. It contains six sub-datasets, i.e., (C_1, C_2, …, C_6), and each sub-dataset records 7 sensor measurements, that is, a cutter's milling force, vibration signals on X, Y and Z directions, and the RMS value of the acoustic emission signal. Only sub-datasets C_1, C_4, and C_6 have the corresponding flank wear as the label. Following the settings in [87], we transfer the tool wear prediction task to the tool RUL prediction task in this benchmark study.

1) Self-collected datasets

XJTUSpurgear Dataset. The experiment platform is depicted in Fig. 20(a), which consists of a driving motor, a belt, a shaft, and a gearbox. Among them, the type of the motor is an AC variable frequency motor, and its power supply is a single-phase alternating current (220 V, 60/50 Hz). Twelve 1D-accelerometers (PCB333B32) are mounted on the gearbox to collect the vibration signals, and the signals of the first sensor are used. In the experiment, four types of root cracks with different crack degrees are prefabricated on the spur gear, as shown in Fig. 20(b). Together with the normal state, a total of five kinds of vibration signals are collected. Three different speeds are simulated, that is, 900 r/min, 1200 r/min, and variable speeds from 0 to 1200 r/min to 0. Besides, the sampling frequency is set to 10 kHz during the experiments. In our benchmark study, the data of the former two working conditions is used, and we regard the data collected under different working conditions as different failure modes, so this task becomes a 10-class classification task.

XJTUGearbox Dataset. The experiment platform is depicted in Fig. 21(a), which is composed of a driving motor, a controller, a planetary gearbox, a parallel gearbox and a brake. Among them, the type of the motor is a 3 Phase and 3 HP motor, and its power supply is a three-phase alternating current (230 V, 60/50 Hz). Two 1D-accelerometers (PCB352C04) are mounted on the X and Y directions of the planetary gearbox to collect the vibration signals, and the signals in the Y direction are used. In the experiment, four types of planetary gear failure modes and four types of bearing failure modes are prefabricated on the planetary gearbox. As shown in Fig. 21(b), the gear failures include the tooth surface wear, missing tooth, root cracks and tooth broken. The bearing failures include the ball fault, inner race fault, outer race fault and a mixed fault of the aforementioned three bearing faults. Therefore, together with the normal state, a total of nine kinds of vibration signals are collected. Besides, the motor speed is set to 1800r/min and the sampling frequency is set to 20480 Hz during the experiments.

2) Open source datasets

There are many famous open source datasets in the field of intelligent fault diagnostics and prognostics. In our benchmark study, six open source datasets are used, including the CWRU dataset, the PU dataset, the MFPT dataset, the SEU dataset, the CMPASS dataset and the PHM2010 dataset. The former four datasets are used for fault diagnosis, and the last two datasets are utilized for prognosis, i.e., remaining useful life (RUL) prediction. The detailed descriptions of these open source datasets are shown in Table 2.

4. Benchmark study

4.1. Experiment setup

1) Data preprocessing
A. Data preprocessing for fault diagnosis

In this experiment, the aforementioned six fault diagnosis datasets are all normalized by Min-max normalization, and the effect of

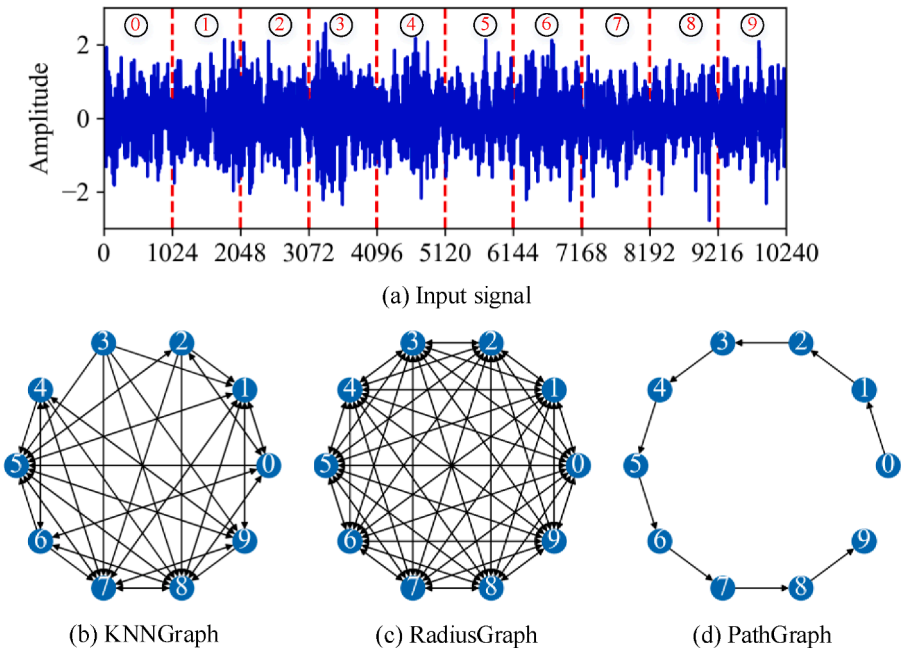


Fig. 22. Examples of graph construction using three graph construction methods.

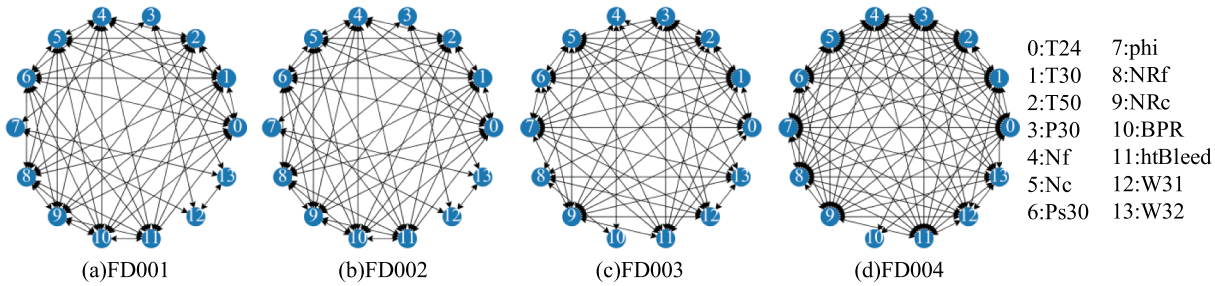


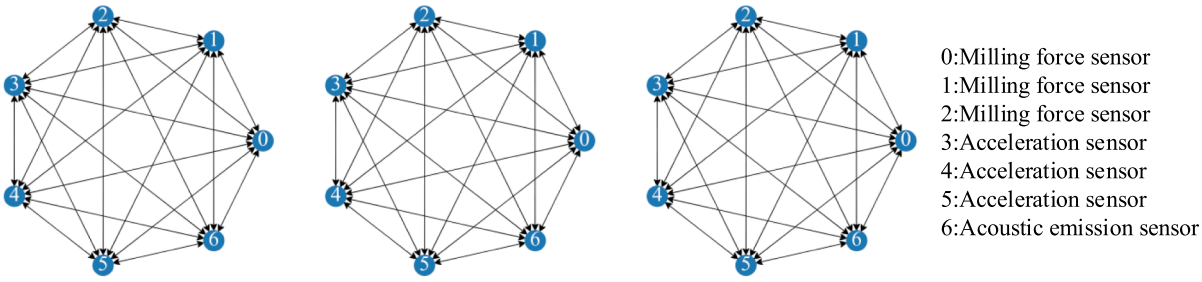
Fig. 23. The constructed sensor network of CMAPSS dataset.

the data normalization methods on the results can be found in [88]. Then, a sliding window of length 1024 is leveraged to truncate the raw signals without overlap. After the data splitting, 1000 subsamples are obtained for each dataset, and 80% of them are randomly selected as the training set, and the other 20% are reserved as the test set. After that, the three graph construction methods proposed in Section 3 are utilized to construct graph data by using subsamples in the training set and test set, respectively. In the process of graph construction, we use every 10 subsamples to construct one graph data. Therefore, the final training set contains 80 sub-graphs and the test set contains 20 sub-graphs. Besides, two commonly used input types are considered, namely the time domain input (TD) and frequency domain input (FD), where the frequency domain input means the subsample is preprocessed by FFT (Fast Fourier Transform).

In order to better understand the difference between the graphs constructed by the three graph construction methods, we give an example here. The input signal of this example, as shown in Fig. 22(a), contains 10240 data points, which is taken from the normal state of the XJTUGearbox dataset. Before applying the three graph construction methods, we first need to divide the input signal into several subsamples according to Eq. (23), where the length d of each subsample is set to 1024. Then, 10 samples can be obtained, which are sequentially numbered from 0 to 9. After that, we can use the three graph construction methods to find the neighbors of each subsample, and the constructed graphs are shown in Fig. 22. As can be seen, in Fig. 22(b), 5 neighbors of each subsample are found, and in Fig. 22(c), the neighbors within the radius ($\epsilon = 0$) of each subsample are found, and in Fig. 22(d), each subsample is connected in chronological order.

B. Data preprocessing for prognosis

CMAPSS dataset. The data preprocessing of CMAPSS dataset follows the settings in [89], where only 14 sensor measurements are



(a) C_1 (b) C_4 (c) C_6

Fig. 24. The constructed sensor network of PHM2010 dataset.

Table 3
The detailed structure of the proposed framework.

Node level		Graph level	
Layer	Filter	Layer	Filter
Input	/	Input	/
GConv_1	1024×1024	GConv_1	1024×1024
BatchNorm_1	1024	BatchNorm_1	1024
ReLU_1	/	ReLU_1	/
		GPool_1	1024
		Readout_1	Mean pool
GConv_2	1024×1024	GConv_2	1024×1024
BatchNorm_2	1024	BatchNorm_2	1024
ReLU_2	/	ReLU_2	/
		GPool_2	1024
		Readout_2	Mean pool
FC_1	1024×512	FC_1	1024×512
Dropout ratio	0.2	Dropout ratio	0.2
FC_2	$512 \times C$	FC_2	$512 \times C$ or 512×1

used, i.e., 2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 and 21. The linear degradation model is applied to calculate the RUL labels and the maximum RUL of each engine is set to 125. Since FD002 and FD004 contain data that collected under multiple working conditions, k-means clustering is used to cluster the dataset according to the working conditions, and then each cluster is normalized by Min-max normalization. Then, the multiple sensors are constructed as sensor networks by using RadiusGraph, as shown in Fig. 23. The spatial–temporal graphs can be generated by assigning node features at different time stamps to the sensor networks, and to get enough training samples, the window length takes 30 at this time.

PHM2010 dataset. In this benchmark study, we turn the tool wear problem into a tool RUL prediction problem, and the RUL of each milling cutter can be defined by Eq. (29). The data preprocessing of the PHM2010 dataset follows the settings in [87], only the sub-datasets C_1, C_4 and C_6 are utilized. The seven sensor measurements are also normalized by Min-max normalization, and the process of sensor network construction and graph generation is consistent with the previous dataset, and the constructed sensor network is shown in Fig. 24. To give a comprehensive benchmark result, the three-fold strategy is adopted, which means that when two of the three datasets are used for model training, the other dataset will be used for model testing.

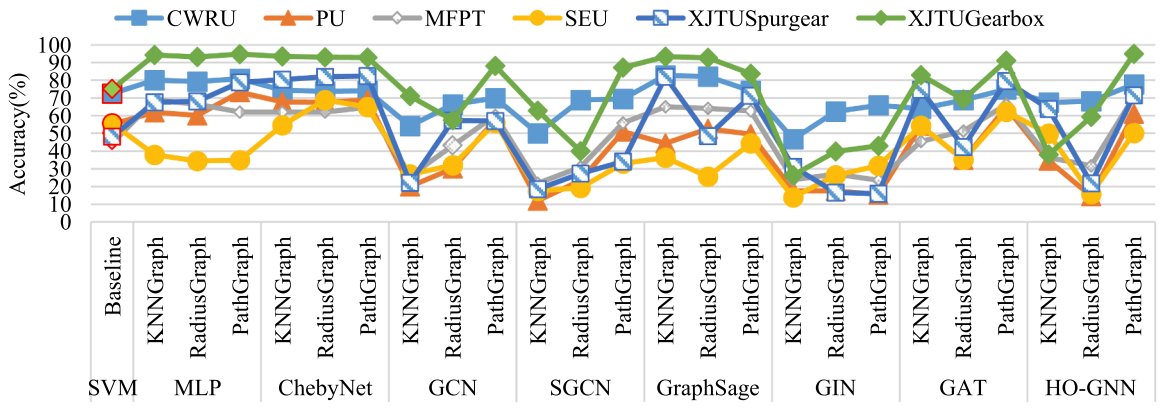
$$\mathrm{RUL}(t) = t_w - t \tag{29}$$

where t denotes the current time, and $\mathrm{RUL}(t)$ indicates the assigned RUL of this time. t_w represents the time when the tool wear reaches the failure threshold, and the threshold sets to 0.16 mm in this paper.

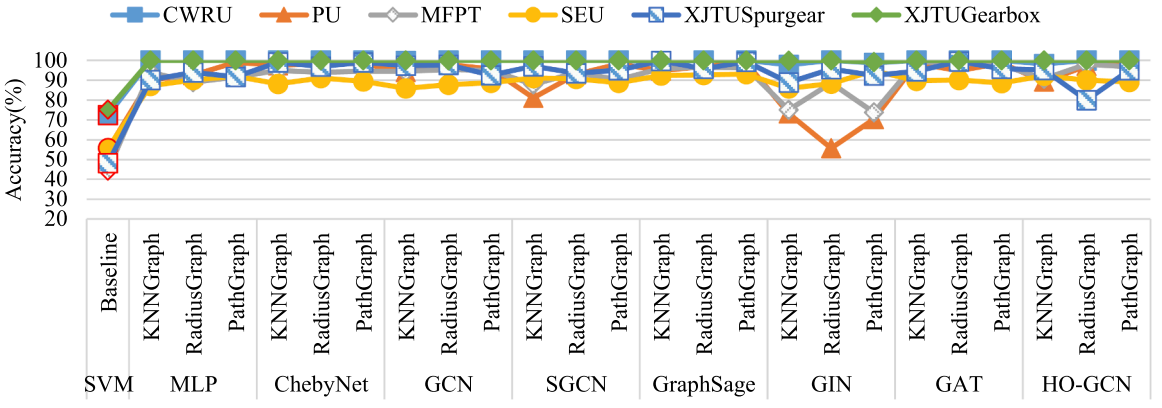
2) Detailed model structure

The detailed model structure of the proposed framework is listed in Table 3. In this table, the GConv layer can be any of the seven GNNs mentioned, and the GPool layer can be any of the four graph pooling layers discussed before. Besides, it can be found from this table that except for the graph pooling layer and the readout layer, the parameters of the other filters in the node level and graph level architectures are consistent. Such definition makes the dimensions of the extracted features of each model consistent and ensures the fairness of comparison.

In this benchmark study, apart from the aforementioned seven GNNs with four kinds of graph pooling methods are explored, the multilayer perceptron (MLP) is also utilized for comparison, where the MLP model can be obtained by replacing the GConv layer in



(a) Time domain input



(b) Frequency domain input.

Fig. 25. The experimental results of node-level fault diagnostics on each dataset.

Table 3 with a fully connected layer. In addition, a well-known traditional fault diagnosis method, a support vector machine (SVM) with RBF kernel [90], is also adopted to obtain baseline results.

3) Training strategy and evaluation metric

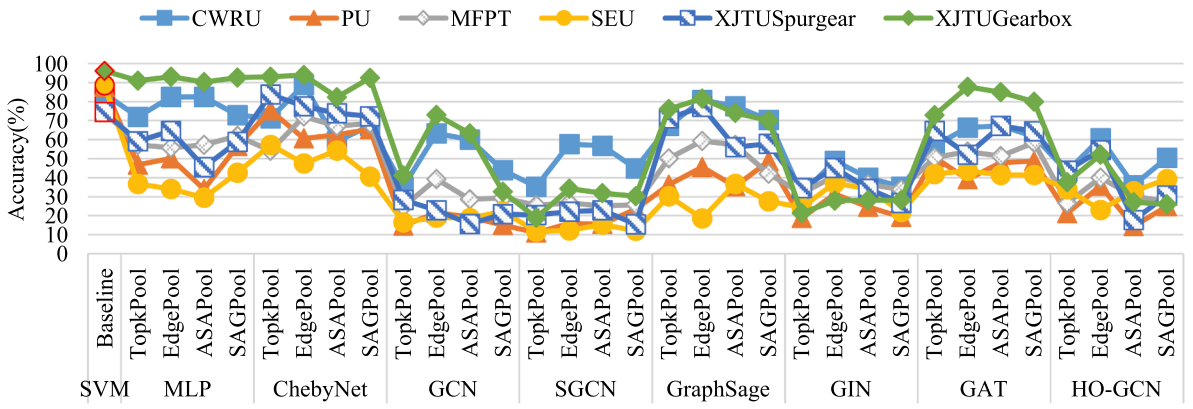
The initial learning rate is 0.01 and stochastic gradient descent (SGD) with momentum is used as the optimizer, where the momentum of SGD is 0.9. The batch size is 64 and each model is trained with 100 epochs for fault diagnosis, while for prognosis, each model is trained with 180 epochs. The learning rate decay strategy is also used to adjust the learning rate, and the weight decay value is initialized to 0.0005.

In fault diagnosis, the overall accuracy is utilized as the evaluation metric, which is defined as the ratio of the number of correctly classified samples to the total number of samples. In prognosis, the root mean squared error (RMSE) and score function (SF) are leveraged as the evaluation metric, where SF is defined as follows:

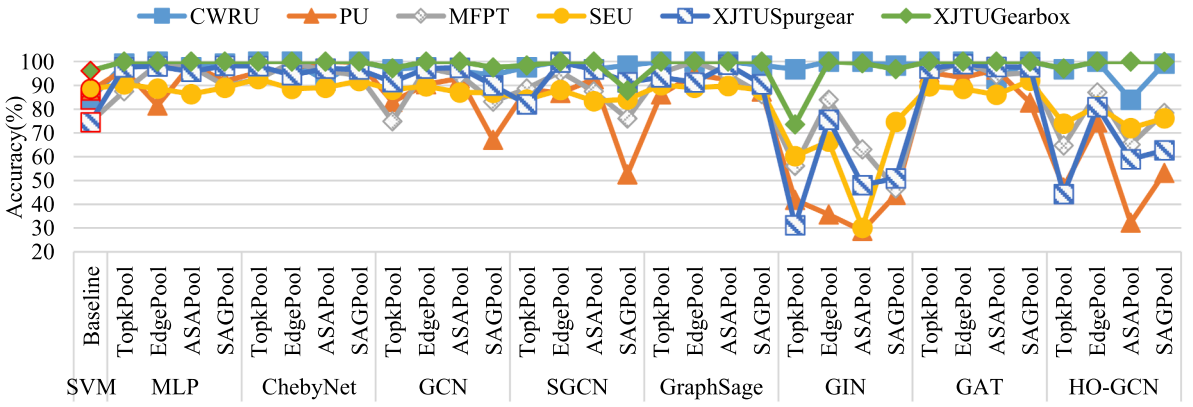
$$\mathrm{SF}=\sum_{i=1}^{\Psi_G}\mathrm{SF}_i,\quad \text{with}\quad \mathrm{SF}_i=\begin{cases} e^{-\frac{\delta_i}{13}}-1, & \delta_i<0 \\ e^{\frac{\delta_i}{13}}-1, & \delta_i\geqslant 0 \end{cases} \tag{30}$$

where SF is the calculated score. $\delta_i=\overline{\mathrm{A}}_i-\mathrm{A}$ is the error between predicted RUL and the ground-truth RUL, and Ψ_G denotes the constructed graph set.

Besides, in order to reduce the randomness of the results, each experiment is repeated five times, and the average value of five test results is taken as the final result.



(a) Time domain input



(b) Frequency domain input.

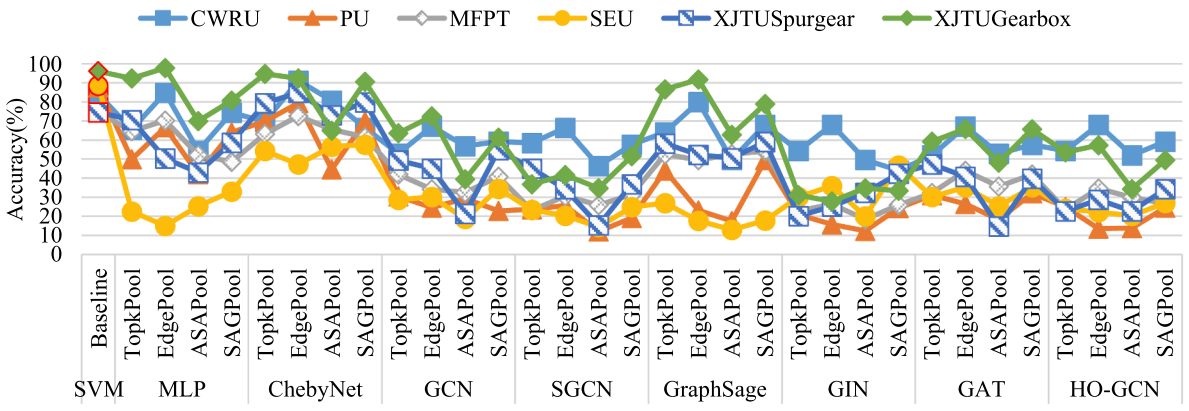
Fig. 26. The experimental results of graph-level fault diagnostics with KNNGraph topology.

4.2. Experimental results of node-level fault diagnostics

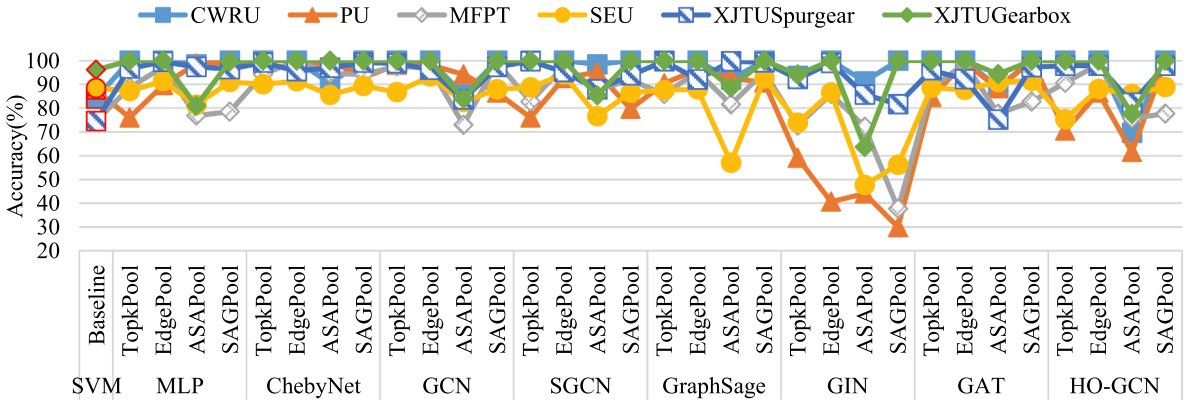
In node-level fault diagnostics, three types of graph construction methods and two different input types on model results are discussed. Since the feature length of each subsample is too large, the curse of dimensionality problem will happen in SVM. Therefore, in order to avoid such problems in fault diagnosis, ten statistical features are extracted from each subsample [91] as the new features of these subsamples, including mean value, peak value, root mean square, standard deviation, skewness, kurtosis, waveform factor, peak factor, pulse factor, and margin factor. The experimental results are shown in Appendix Table A1, and its corresponding line graph is shown in Fig. 25.

4.3. Experimental results of graph-level fault diagnostics

In graph-level fault diagnostics, several samples are selected and constructed as a graph, and the constructed graph shares the same label with the selected samples. By using the proposed graph-level fault diagnostics architecture, the label of each graph can be predicted. As aforementioned in the previous experiments, we extract ten statistical features for each subsample as the input of SVM. Here, we also extract ten statistical features for each node feature of a graph, and then we use sum operation to obtain the final manual features of a graph and utilize it as the input of SVM. The experimental results are shown in Appendix from Table A2, Table A3, Table A4, and their corresponding line graphs are shown in Fig. 26 to Fig. 28.Fig. 27.



(a) Time domain input



(b) Frequency domain input

Fig. 27. The experimental results of graph-level fault diagnostics with RadiusGraph topology.

4.4. Experimental results of graph-level prognostic

In this section, two prognosis datasets, i.e., CMAPSS dataset and PHM2010 dataset, are utilized to obtain the RUL prediction benchmark results. In this experiment, since the length of each subsample is relatively small, we directly use the sum operation to get the manual features for SVM. The experimental results are shown in Appendix Table A5, and its corresponding line graph is shown in Fig. 29.

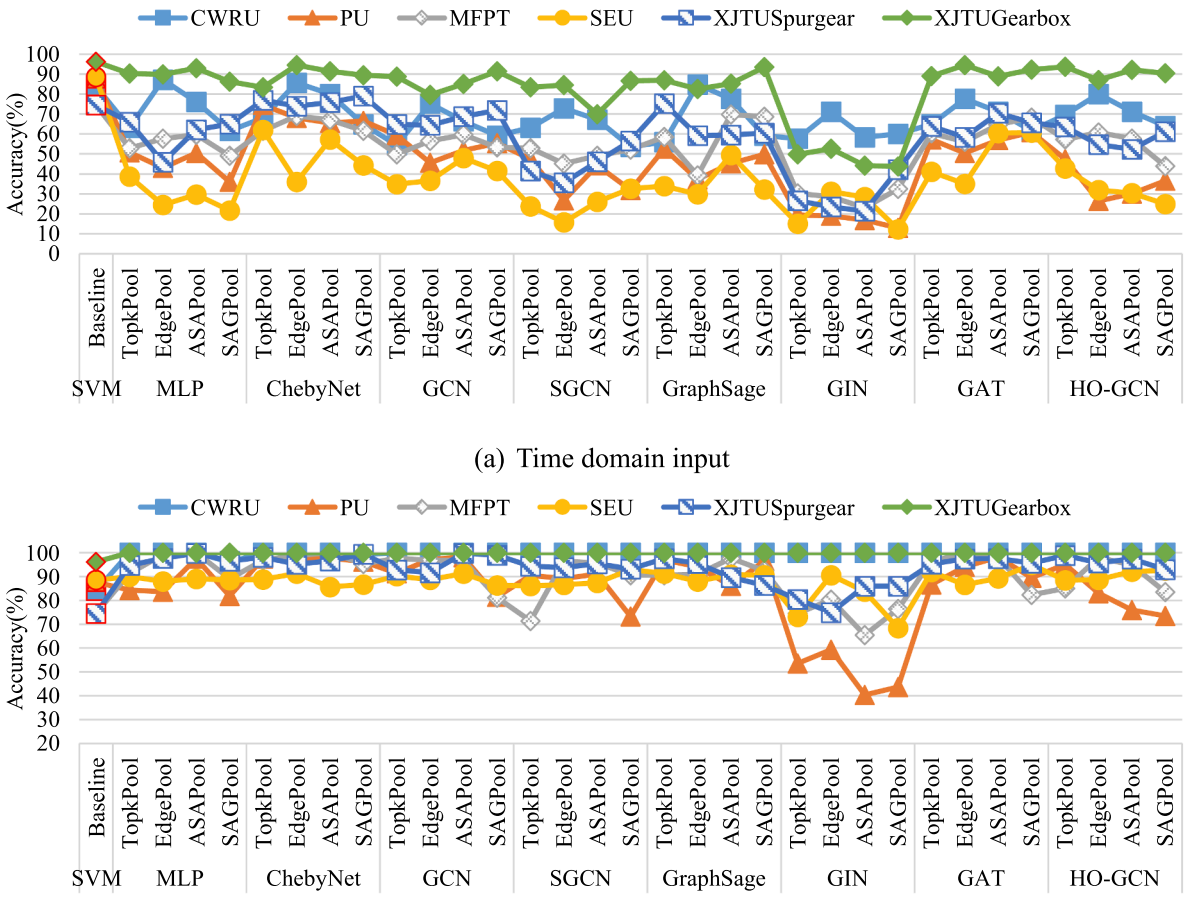
4.5. Benchmark results analysis

1) The influence of input types

As shown in these results, whether it is node-level or graph-level fault diagnosis, after using frequency domain input, the performance of each model has been greatly improved. Especially on the CWRU and XJTUGearbox datasets, the accuracy of most models approaches 100%. This is because the influence of environmental noise on node features has been reduced after FFT, which makes it easier to learn meaningful node or graph representations.

2) The influence of graph topologies on model performance

It is found from Fig. 25 and Fig. 28 that when using KNNGraph and RadiusGraph topologies, the diagnostic accuracies at graph-level are lower than those at the node-level. However, when using PathGraph topology, as shown in Fig. 28, the diagnostic accuracies of each model are greatly improved compared to node-level classification. Here, PathGraph is the intuitive representation of the univariate time series, so the models with PathGraph topology can perform better. These results indicate that the topology of a graph as



(a) Time domain input

(b) Frequency domain input

Fig. 28. The experimental results of graph-level fault diagnostics with PathGraph topology.

a prior knowledge can influence the model performance. Therefore, it is very important to choose a suitable topology when performing graph-level fault diagnostics.

3) The influence of graph pooling methods on model performance

The performance of the seven GNNs with four graph pooling methods is discussed, and in order to explore the influence of graph pooling methods on model performance, we draw the radar chart of the graph-level fault diagnosis results of two best performance models (i.e., ChebyNet and GraphSage) in Fig. 30.

It can be observed from this figure that both two models with EdgePool achieve the best performance on most datasets, while models with TopkPool perform the worst. The reason for these results may be that EdgePool can integrate the edge features between two nodes into the generated new node features through edge contraction, and such a graph pooling method immediately and naturally takes the graph structure into consideration and guarantees that no nodes are completely dropped. This also makes EdgePool different from other pooling methods that use Top-k or attention mechanisms to calculate node scores and discard nodes.

4) The best performance model

Since the benchmark results of each model on the eight datasets are obtained, in order to answer the question of which model has the most stable performance on each dataset and give a practical guide to users, we report the model that performs best under different tasks with different graph construction methods. The evaluation metric is the mean value of the model's diagnostic or prognostic results of different datasets, and the model performance evaluation results are shown in Fig. 31.

From these results, it can be found that when performing node-level fault diagnostics, ChebyNet with PathGraph topology can achieve the best results under time domain input, while GraphSage with PathGraph topology can obtain the best results under

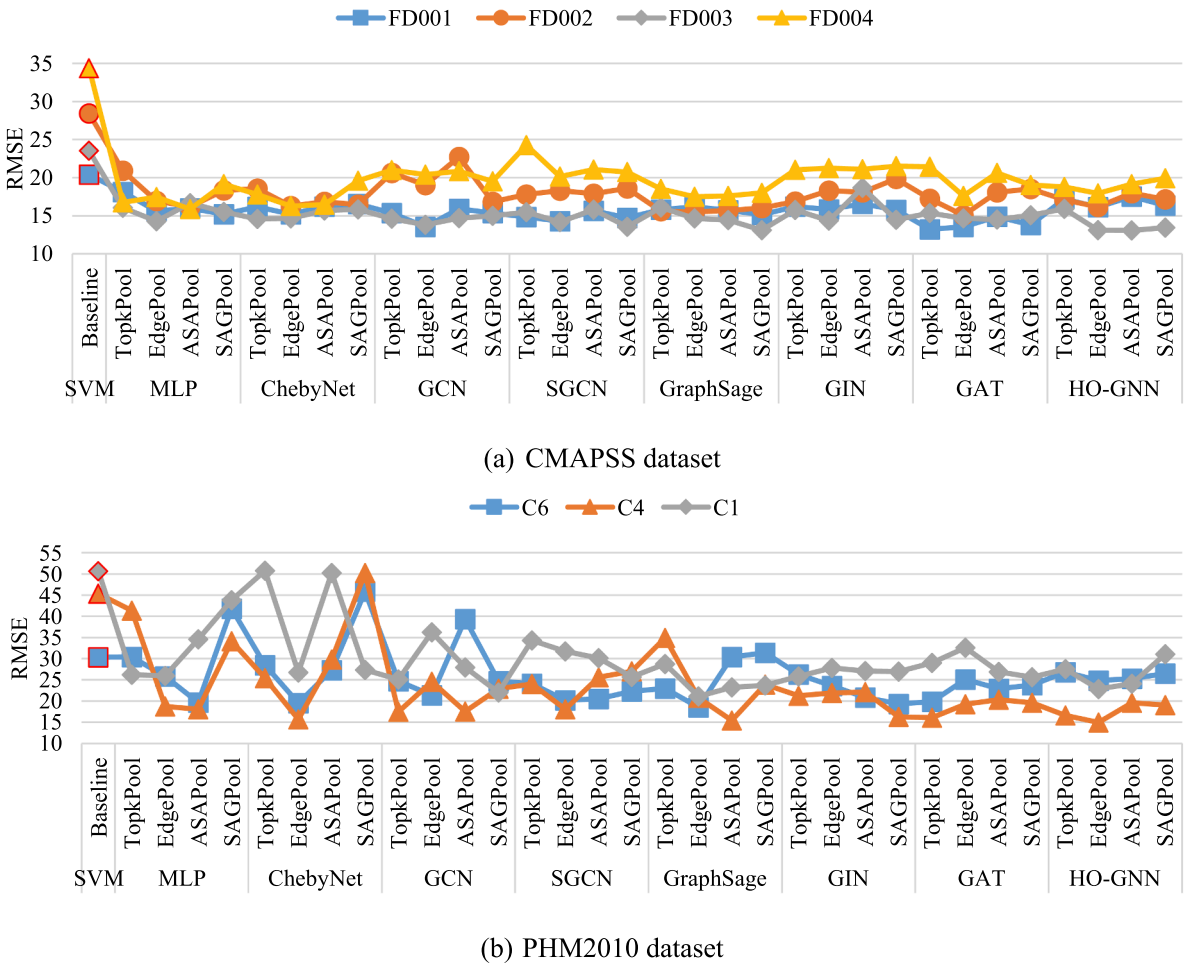


Fig. 29. The experimental results of graph-level RUL prediction.

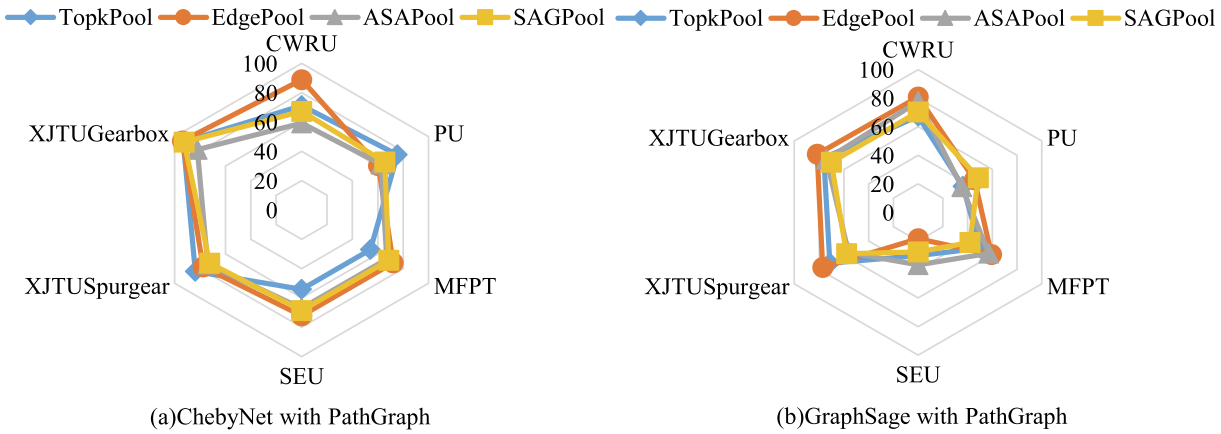


Fig. 30. The influence of graph pooling method on model performance.

frequency domain input. In graph-level fault diagnostics, ChebyNet with EdegPool can acquire the best results under time domain input, while MLP with ASAPool can get the best results under frequency domain input. In graph-level prognostics, GAT with EdgePool can achieve the smallest RMSE in the CMAPSS dataset, while GraphSage with EdgePool can achieve the smallest RMSE in the PHM2010 dataset. Moreover, it can be observed that the node-level fault diagnostic architecture performs more stable than the graph-

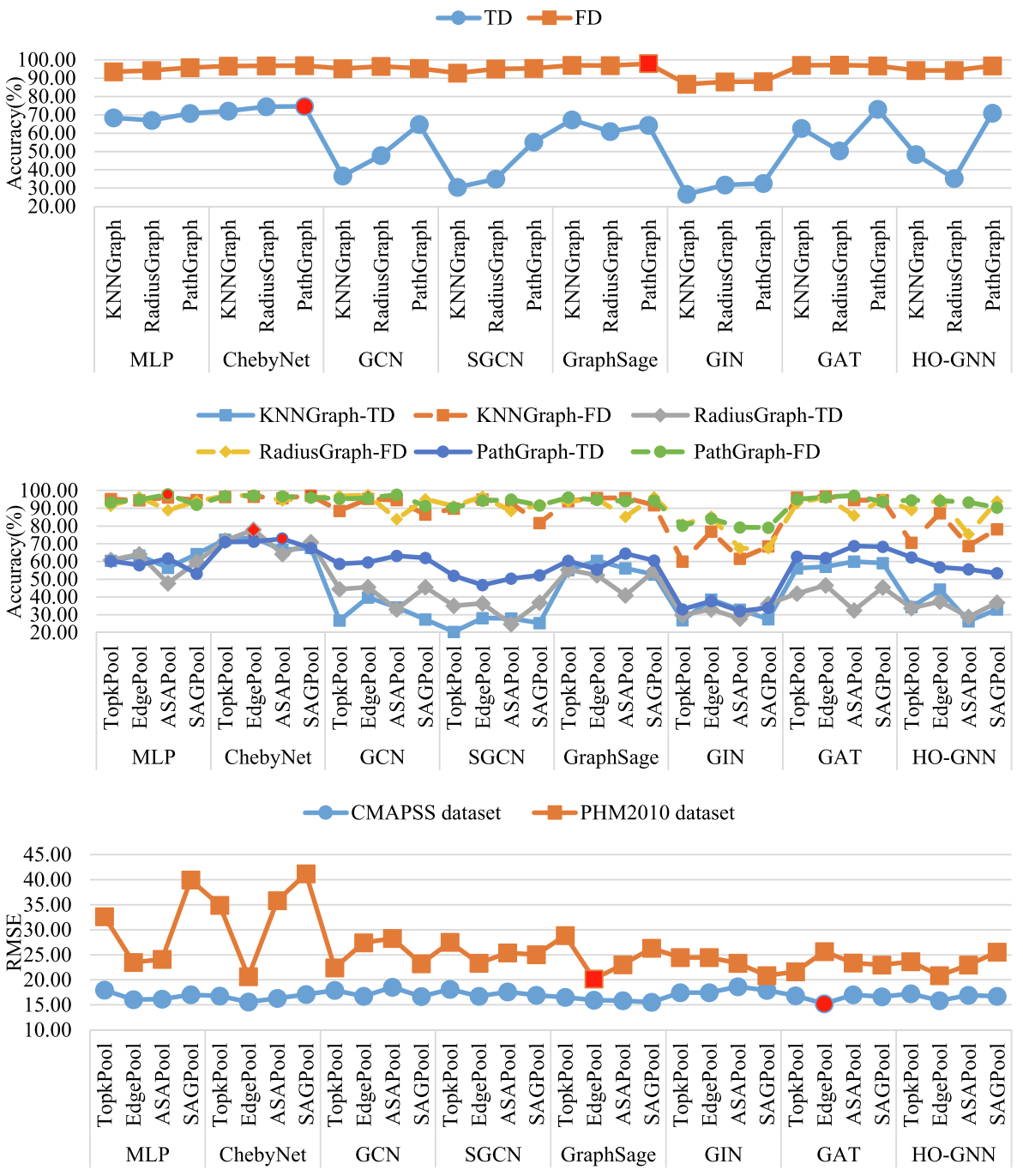


Fig. 31. Model performance evaluation results.

level fault diagnostic architecture.

Therefore, from the above analysis, it can be found that the overall performance of ChebyNet on these eight datasets is best. Using ChebyNet and EdegPool at the same time can also achieve the best results in these two kinds of graph-level tasks. Besides, for fault diagnosis, adopting the node-level fault diagnostic architecture could achieve better model performance, and in graph-level fault diagnostic, models with PathGraph topology can obtain better results.

Table 4
GCN vs. MLP on XJTUSurgear and XJTUGearbox datasets (%).

Dataset	One-layer MLP	Two-layer MLP	One-layer GCN	Two-layer GCN
XJTUSurgear	22.65	51.21	32.84	55.59
XJTUGearbox	23.67	84.42	34.38	87.14

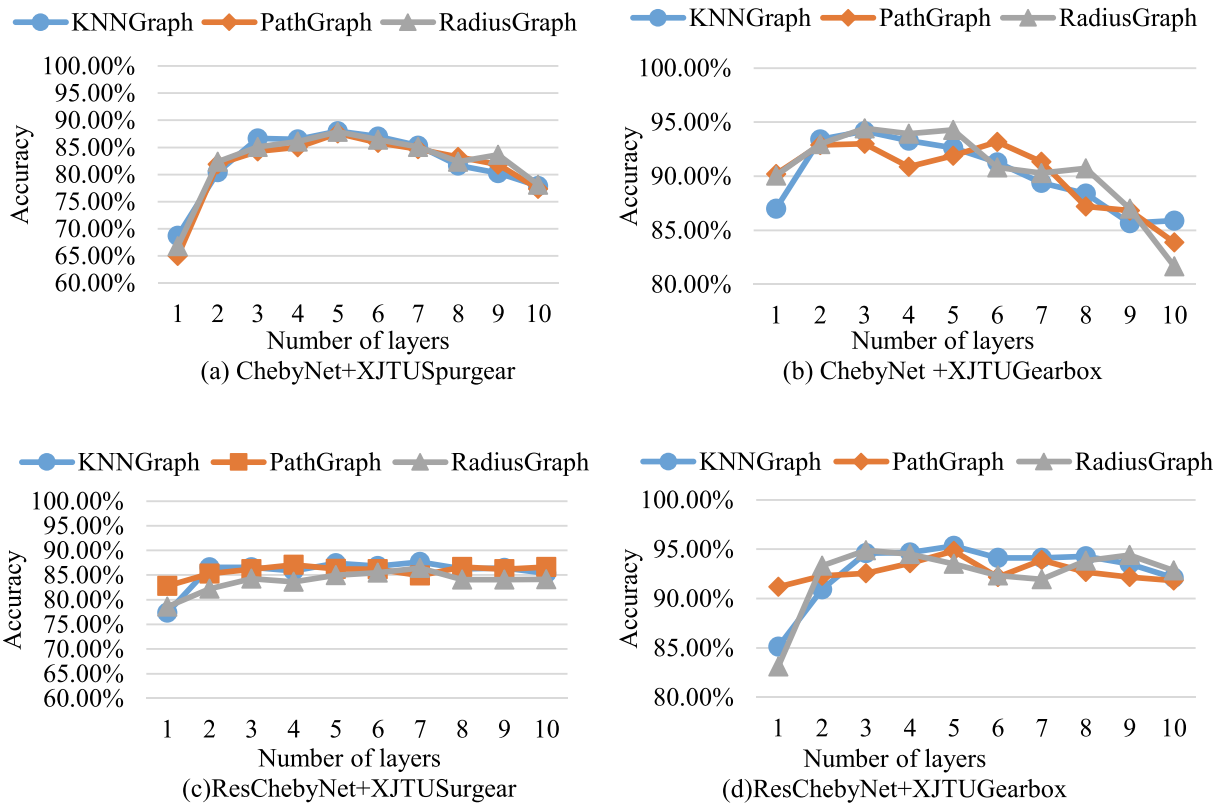


Fig. 32. The model performance with different number of layers.

5. Discussion

In this section, we will discuss four issues related to GNNs, that is, why GCNs work, can GCNs go deep, how the aggregators influence model performance and the robustness of GNNs. In addition, the experiments in this section will only use time domain input for convenience.

5.1. Why do GCNs work?

To understand the reasons for why GCNs perform better than MLP in most cases, we start the comparison from their propagation rule, where the propagation rule of MLP is defined as:

$$\boldsymbol{H} = \sigma(\boldsymbol{X}\boldsymbol{\Theta}) \tag{31}$$

It can be observed from Eq. (7) and Eq. (31) that the only difference between a GCN and MLP is the graph convolution matrix $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$applied on the left of the node feature matrix $\mathbf{X}$, and this also indicates that the graph convolution matrix plays the critical role in the huge performance gain. In order to observe the influence of graph convolution, we conduct a simple experiment on the XJTUSurgear and the XJTUGearbox datasets. In this node-level fault diagnosis, only PathGraph is used as the input of GCN and MLP, and the results are shown in Table 4. From these results, it can be seen that GCN can achieve the best performance in these two datasets, and the performance of a one-layer GCN is much better than a one-layer MLP, which validates the effectiveness of graph convolution.

To clearly examine the graph convolution matrix, we introduce the matrix form of Laplacian smoothing [92], which is defined as:

$$\widehat{\boldsymbol{X}} = \boldsymbol{X} - \gamma\boldsymbol{D}^{-1}\boldsymbol{L}\boldsymbol{X} \;=\; (\boldsymbol{I}_n \text{ - } \gamma\boldsymbol{D}^{-1}\boldsymbol{L})\boldsymbol{X} \tag{32}$$

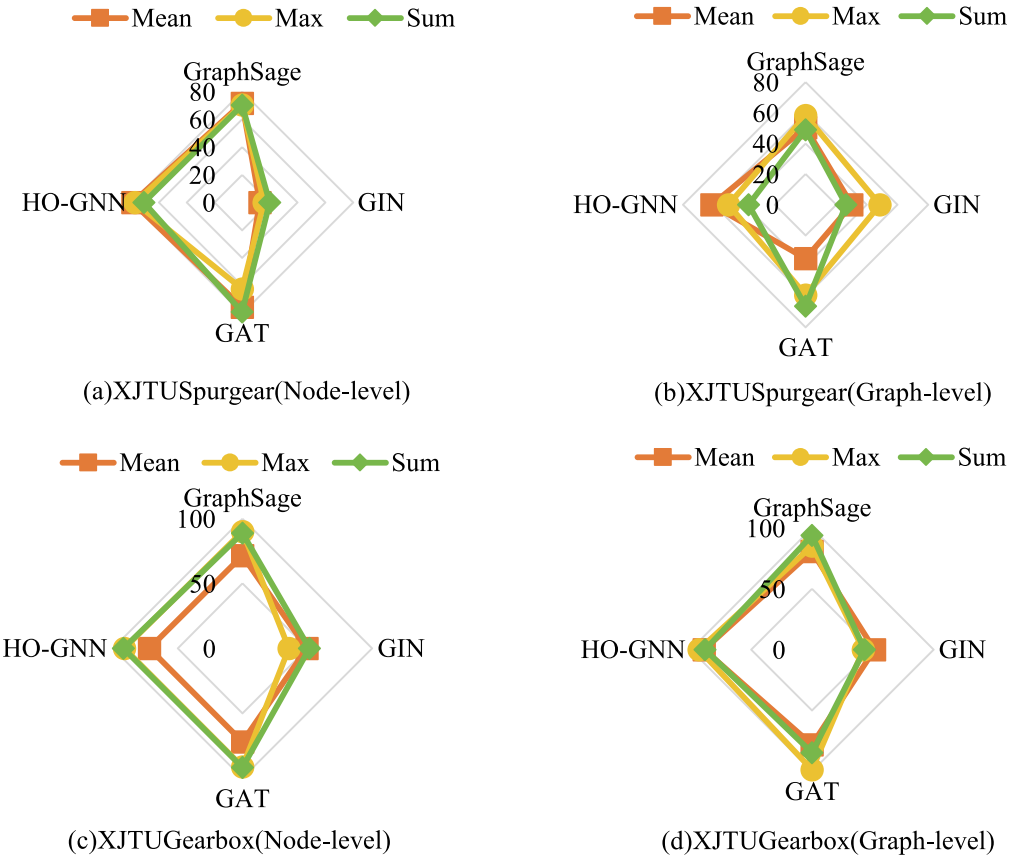


Fig. 33. The influence of aggregator on model performance.

where $0 < \gamma \leqslant 1$ is a balance parameter which controls the weight between the features of the current node and its neighbors, and $\widehat{\boldsymbol{X}}$ is the new node feature matrix. By taking $\gamma = 1$, the above equation becomes $\widehat{\boldsymbol{X}} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{X}$, which means only neighbors' node features are used in Laplacian smoothing.

Therefore, in Eq. (32), if we take $\gamma = 1$ and replace $\boldsymbol{D}^{-1}\boldsymbol{L}$ with the symmetrically normalized Laplacian$\boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$, we can have $\widehat{\boldsymbol{X}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\boldsymbol{X}$, which is exactly the graph convolution in Eq. (7). Thus, we can call the graph convolution a special form of Laplacian smoothing. In addition, it is also possible to understand the process by which GCN obtains new node features, that is, through the weighted average of the node itself and its neighbors. Since the nodes in the cluster tend to be densely connected, the smoothing process makes their node features become similar, which makes the subsequent node classification task become easier.

5.2. *Can GCNs go as deep as deep neural networks?*

Compared with shallow networks, the superiority of deep neural networks is that they can mine useful information from raw data by establishing an approximate complex nonlinear function. As mentioned in the previous section, Laplacian smoothing of GCN makes the node features become similar in the cluster. However, a GCN with multiple layers will repeatedly apply Laplacian smoothing to different clusters, which will lead to an over-smoothing problem and make the nodes indistinguishable [93].

To illustrate this phenomenon, we increase the number of layers of ChebyNet from 1 to 10. The experiments are carried out on XJTUSpurgear and XJTUGearbox datasets for node-level fault diagnostics, and all three types of graph topologies are tested and only the time domain node features are considered. The experimental results are shown in Fig. 32(a) and Fig. 32(b). It can be observed that as the number of layers increases, the performance of ChebyNet first raises and then dramatically drops. The best results are obtained when the number of layers takes 2 to 5.

Fortunately, the residual connection provides a possible way to avoid such problem. Here, ChebyNet with residual connection (ResChebyNet) follows the ordering: GraphConv → Normalization → ReLU → Addation, and the experimental results are shown in Fig. 32(c) and Fig. 32(d). It can be found that the performance of ChebyNet gradually stabilizes rather than declines as the number of network layers increases. This result indicates that residual connection does help to overcome the over-smoothing problem, and more approaches can be found in Ref. [94].

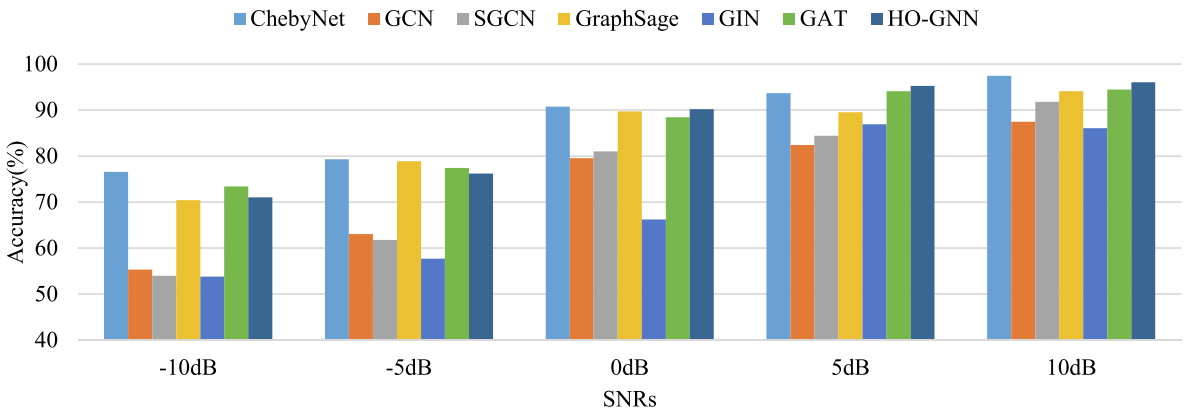


Fig. 34. Experimental results of GNNs under different noise levels.

5.3. How the aggregators influence model performance

As discussed in Section 2.3, the mean aggregator, max aggregator and sum aggregator are three widely used AGGREGATE(·) functions for the spatial-based GCNs. To investigate the influence of aggregator on model performance, we conduct fault diagnosis experiments on the XJTUSupergear and the XJTUGearbox datasets, and change the AGGREGATE (·) functions of the four spatial-based GCNs. In the experiments, the PathGraph topology is utilized and the EdgePool is adopted for graph-level fault diagnosis, and the experimental results are shown in Fig. 33.

It can be found that the performance of the four spatial-based GCNs fluctuates with the change of the aggregator, and this also makes it difficult to choose a suitable aggregator so that each model can achieve the best performance. However, by counting the performance of each model in different aggregators, we can find that compared to the model using other aggregators, the models using the sum aggregators can obtain relatively better diagnostic results in this experiment. This is because the sum aggregator can use the full features of a node's adjacent nodes to learn accurate structural information. Therefore, we recommend the sum aggregator as the first choice when choosing an aggregator.

5.4. Robustness of GNNs

Since the robustness of the model in fault diagnosis has always been a topic of great concern, in this section we will also discuss the robustness of GNNs. In order to explore the robustness of GNN to noise, we implement node-level fault diagnosis on the XJTUGearbox dataset and only use the PathGraph as model input. In this experiment, according to the settings in [95], the Gaussian white noise will be added to the raw vibration signals with different signal-to-noise ratios (SNRs), where the definition of SNR is shown in Eq. (33) and it takes from −10 dB to 10 dB in this study. The experimental results are shown in Fig. 34.

$$\mathrm{SNR}_{dB} = 10log_{10}\left(\frac{p_{\mathrm{singal}}}{p_{\mathrm{noise}}}\right) \tag{33}$$

where p_{singal} and p_{noise} are the power of the raw signal and the noise, respectively.

It can be observed that the performance of GNNs decreases with the increase of noise, and increases with the decrease of noise. Besides, we also find that the performance of GIN is greatly improved when the SNR exceeds 0 dB. This result indicates that the anti-noise ability of GNNs still needs further improvement, and increasing the SNR may be a possible way to enhance the model performance.

6. Further directions

6.1. New paradigm for realizing graph convolution

Existing GCNs can be regarded as message passing based approaches, and the message passing mechanism refers to obtaining the new node features of a node by aggregating features of its neighbors according to a certain rule [96]. This mechanism pushes representations of adjacent nodes closer to each other, and thereby improves the performance of the model in node classification. Theoretically speaking, if we stack graph convolutional layers infinitely, the representation of all nodes will converge to one point. This phenomenon limits the acquisition of deep GCNs. Although the aforementioned methods such as residual connection can alleviate this phenomenon, a new paradigm for realizing graph convolution is still needed to solve this problem.

Table A1
The results of node-level fault diagnostics.

Model	GraphType	CWRU		PU		MFPT		SEU		XJTUSpurgear		XJTUGearbox	
		TD	FD	TD	FD	TD	FD	TD	FD	TD	FD	TD	FD
SVM		72.41		55.15		44.39		55.75		48.10		75.06	
MLP	KNNGraph	80.00	**100.00**	61.91	90.18	**68.82**	94.00	37.98	87.19	67.61	90.15	**94.19**	**100.00**
	RadiusGraph	79.20	**100.00**	60.21	92.52	**66.55**	88.98	34.41	90.11	68.00	93.96	**93.22**	**100.00**
	PathGraph	**80.80**	**100.00**	**73.23**	99.02	62.04	91.61	35.00	91.96	78.75	91.75	94.79	**100.00**
ChebyNet	KNNGraph	74.40	**100.00**	**67.81**	97.51	62.00	94.86	**54.63**	88.17	80.44	99.01	93.40	**100.00**
	RadiusGraph	73.76	**100.00**	**67.51**	**98.37**	62.08	93.96	**68.81**	91.18	**81.91**	96.89	92.97	**100.00**
	PathGraph	74.08	**100.00**	68.80	98.34	64.67	94.51	**64.85**	89.40	**82.37**	98.95	92.90	**100.00**
GCN	KNNGraph	54.16	99.76	20.09	94.06	25.92	94.55	26.93	85.83	22.22	97.27	71.09	99.94
	RadiusGraph	66.40	**100.00**	30.16	97.85	43.14	95.29	32.05	87.70	57.40	97.70	57.56	**100.00**
	PathGraph	69.92	**100.00**	57.69	95.60	60.16	94.94	55.99	88.66	56.95	92.54	88.10	**100.00**
SGCN	KNNGraph	50.00	**100.00**	12.13	81.14	21.84	87.92	17.63	91.18	18.59	97.00	62.79	99.99
	RadiusGraph	68.80	**100.00**	23.38	92.92	30.98	93.73	19.35	90.65	27.30	93.50	40.00	**100.00**
	PathGraph	69.44	**100.00**	50.99	98.12	55.88	90.27	33.04	88.78	34.19	95.00	87.12	**100.00**
GraphSage	KNNGraph	**82.80**	99.20	44.46	94.80	65.02	95.92	36.39	**92.39**	**81.93**	**99.75**	93.29	**100.00**
	RadiusGraph	**82.08**	**100.00**	52.65	96.89	64.04	96.43	25.65	**92.60**	48.56	95.75	92.71	**100.00**
	PathGraph	74.00	**100.00**	49.60	98.80	62.67	96.20	44.43	93.15	71.15	**99.35**	83.71	**100.00**
GIN	KNNGraph	46.72	97.76	17.57	73.20	23.81	75.02	13.99	85.93	31.18	88.81	26.34	99.89
	RadiusGraph	62.40	**100.00**	17.54	55.69	27.06	88.63	26.45	88.35	16.70	95.70	39.89	**100.00**
	PathGraph	65.84	98.72	15.60	70.37	23.53	73.80	31.84	**94.83**	15.99	92.36	42.98	99.14
GAT	KNNGraph	64.00	**100.00**	54.31	**98.52**	45.57	**99.14**	54.33	89.70	74.76	94.54	82.91	**100.00**
	RadiusGraph	68.80	**100.00**	35.08	94.77	51.37	**98.82**	35.20	90.10	42.40	**99.50**	69.22	**100.00**
	PathGraph	74.88	**100.00**	64.71	98.61	66.31	**97.65**	62.15	88.54	79.60	95.72	91.14	**100.00**
HO-GNN	KNNGraph	67.44	98.24	34.68	89.26	36.39	90.74	49.98	92.18	63.83	95.19	38.16	**100.00**
	RadiusGraph	68.16	**100.00**	14.64	97.14	31.76	98.04	15.82	89.95	21.91	80.00	59.41	**100.00**
	PathGraph	77.76	**100.00**	61.08	**99.69**	**70.67**	96.82	50.29	89.14	71.46	95.02	**94.84**	**100.00**

*TD means time domain input, and FD means frequency domain input.

Table A2
The results of graph-level fault diagnostics with KNNGraph topology.

Model	Pooling Layer	CWRU		PU		MFPT		SEU		XJTUSpurgear		XJTUGearbox	
		TD	FD	TD	FD	TD	FD	TD	FD	TD	FD	TD	FD
SVM		84.00		87.69		74.51		88.50		74.50		96.11	
MLP	TopkPool	**72.00**	99.20	46.77	**96.62**	**57.26**	87.45	36.75	90.50	59.20	97.70	91.00	**100.00**
	EdgePool	82.40	**100.00**	50.15	81.54	55.69	98.82	34.10	88.65	64.60	98.00	93.11	**100.00**
	ASAPool	**82.40**	97.60	33.85	98.77	57.26	**98.04**	29.55	86.35	45.50	95.70	**90.22**	**100.00**
	SAGPool	**72.80**	99.20	56.61	91.69	61.96	89.81	**42.65**	89.05	58.90	**98.20**	**92.56**	**100.00**
ChebyNet	TopkPool	71.20	**100.00**	**75.39**	95.38	54.12	92.55	**57.15**	**92.65**	**83.70**	**98.10**	**93.11**	**100.00**
	EdgePool	**88.80**	**100.00**	**60.61**	**97.23**	**72.16**	99.61	**47.55**	88.55	**77.50**	94.30	**94.00**	**100.00**
	ASAPool	59.20	92.00	**62.46**	**100.00**	**67.06**	96.08	**54.30**	**89.05**	**73.90**	97.10	82.22	**100.00**
	SAGPool	67.20	**100.00**	**65.54**	**99.69**	**68.63**	94.12	40.55	91.90	**72.30**	96.70	92.44	**100.00**
GCN	TopkPool	35.20	96.80	14.77	83.07	24.71	74.90	16.50	88.20	28.30	91.40	41.22	97.22
	EdgePool	63.20	97.60	21.54	90.46	39.22	97.26	19.05	89.60	22.90	97.10	73.00	**100.00**
	ASAPool	60.00	96.00	19.38	92.92	28.63	94.51	18.80	87.00	15.70	97.40	63.22	**100.00**
	SAGPool	44.00	94.40	15.08	67.08	29.41	83.14	22.20	86.90	20.80	90.00	32.56	97.55
SGCN	TopkPool	35.20	97.60	11.08	88.31	25.49	88.63	11.60	84.05	20.30	82.10	18.89	98.33
	EdgePool	57.60	**100.00**	16.00	86.77	26.67	95.69	12.25	88.10	22.10	**99.60**	34.22	**100.00**
	ASAPool	56.80	96.80	15.69	92.61	25.10	87.45	15.35	83.35	22.80	96.70	31.89	**100.00**
	SAGPool	44.80	98.40	23.38	52.61	25.88	76.08	12.05	84.10	15.50	91.30	30.22	87.89
GraphSage	TopkPool	67.20	**100.00**	36.31	86.15	50.19	**95.29**	30.25	90.05	71.30	93.40	76.11	**100.00**
	EdgePool	80.80	**100.00**	45.54	94.46	59.22	**100.00**	18.55	**89.00**	77.20	91.30	81.56	**100.00**
	ASAPool	77.60	**100.00**	35.38	92.31	57.25	94.51	36.70	89.75	56.10	**98.70**	74.00	**100.00**
	SAGPool	70.40	98.40	48.61	87.39	41.96	86.27	27.65	88.00	57.80	90.50	70.11	**100.00**
GIN	TopkPool	31.20	96.80	18.77	41.85	31.76	56.08	24.40	60.30	34.50	31.20	21.33	73.55
	EdgePool	48.80	**100.00**	31.39	35.69	40.78	83.92	37.10	66.40	45.20	75.60	27.89	**100.00**
	ASAPool	40.00	**100.00**	24.62	28.92	36.47	63.14	34.10	30.15	34.20	48.10	28.11	99.33
	SAGPool	35.20	98.40	19.38	44.00	33.72	46.67	22.00	74.65	26.70	50.90	28.11	96.78
GAT	TopkPool	56.80	**100.00**	50.77	95.38	50.59	94.51	41.95	89.60	64.60	97.00	72.78	**100.00**
	EdgePool	66.40	**100.00**	39.08	93.23	53.72	**100.00**	43.70	88.55	52.10	98.90	87.67	**100.00**
	ASAPool	67.20	92.80	48.00	96.61	51.37	94.51	41.40	86.10	67.20	97.80	85.00	**100.00**
	SAGPool	61.60	**100.00**	48.62	82.77	58.43	**95.30**	41.50	**92.00**	64.50	97.70	79.89	**100.00**
HO-GNN	TopkPool	43.20	96.80	21.23	47.08	26.67	64.71	33.95	73.90	43.70	44.30	37.67	97.11
	EdgePool	60.80	**100.00**	35.69	74.46	40.00	87.06	22.95	81.30	54.50	80.90	52.11	**100.00**
	ASAPool	36.00	84.00	14.46	32.31	30.19	65.10	32.90	72.10	17.70	59.00	27.11	**100.00**
	SAGPool	50.40	99.20	24.92	53.23	27.45	78.43	39.30	76.15	30.70	62.70	25.89	99.89

*TD means time domain input, and FD means frequency domain input.

Table A3
The results of graph-level fault diagnostics with RadiusGraph topology.

Model	Pooling Layer	CWRU		PU		MFPT		SEU		XJTUSpurgear		XJTUGearbox	
		TD	FD	TD	FD	TD	FD	TD	FD	TD	FD	TD	FD
SVM		84.00		87.69		74.51		88.50		74.50		96.11	
MLP	TopkPool	65.60	**100.00**	49.85	76.00	64.70	88.63	22.45	87.20	70.50	96.60	92.33	**100.00**
	EdgePool	84.80	**100.00**	66.77	89.54	70.19	97.26	14.90	91.35	50.40	**99.50**	**97.89**	**100.00**
	ASAPool	54.40	98.40	42.15	98.77	52.16	76.86	25.25	81.45	43.00	97.40	70.00	81.00
	SAGPool	**74.40**	**100.00**	64.00	99.08	48.63	78.43	32.95	91.05	58.60	96.40	80.67	99.56
ChebyNet	TopkPool	**69.60**	**100.00**	**69.85**	**98.15**	**63.14**	94.12	**54.30**	**90.10**	**79.40**	99.10	**94.67**	**100.00**
	EdgePool	**91.20**	**100.00**	**79.38**	96.92	**72.55**	**100.00**	**47.20**	91.35	**84.80**	95.60	92.33	**100.00**
	ASAPool	**80.80**	90.40	**44.62**	**100.00**	**65.49**	**94.51**	**56.60**	85.55	**73.10**	97.00	**65.00**	**100.00**
	SAGPool	66.40	**100.00**	**69.85**	92.92	**61.57**	94.12	**57.60**	89.40	**79.80**	**99.10**	**90.56**	**100.00**
GCN	TopkPool	52.80	**100.00**	30.46	**98.15**	41.96	**97.65**	28.65	86.80	49.30	98.90	63.56	**100.00**
	EdgePool	67.20	**100.00**	24.62	98.15	34.12	96.86	30.15	93.30	45.10	96.20	72.45	**100.00**
	ASAPool	56.80	85.60	29.23	94.15	32.55	72.94	18.70	82.65	21.30	83.70	39.44	83.89
	SAGPool	59.20	**100.00**	22.77	86.46	40.78	**99.22**	34.40	87.95	54.80	97.40	61.33	**100.00**
SGCN	TopkPool	58.40	**100.00**	23.69	76.00	23.14	82.75	23.30	88.75	45.10	**99.70**	36.78	**100.00**
	EdgePool	66.40	**100.00**	25.54	92.62	30.59	97.65	20.30	**94.20**	34.30	95.20	41.67	**100.00**
	ASAPool	46.40	98.40	12.00	95.08	25.88	87.84	14.35	76.60	15.20	87.90	34.78	85.33
	SAGPool	57.60	**100.00**	19.08	79.69	31.37	91.77	24.90	86.50	36.80	93.80	51.56	**100.00**
GraphSage	TopkPool	64.00	**100.00**	44.00	90.46	52.55	85.88	26.95	87.55	58.20	**99.70**	86.67	**100.00**
	EdgePool	80.00	**100.00**	23.08	96.31	49.41	95.29	17.55	87.95	52.30	92.10	91.78	**100.00**
	ASAPool	49.60	92.00	17.54	92.61	52.55	81.57	12.95	57.05	50.50	**99.70**	62.78	88.67
	SAGPool	68.00	**100.00**	49.54	90.77	54.12	95.29	17.80	**92.55**	59.00	99.20	78.89	**100.00**
GIN	TopkPool	54.40	93.60	20.92	59.08	22.74	72.55	30.10	73.85	20.20	92.30	31.11	94.22
	EdgePool	68.00	**100.00**	15.69	40.62	26.27	85.88	35.95	86.65	25.10	99.00	27.78	**100.00**
	ASAPool	49.60	91.20	12.31	44.00	18.43	72.16	19.95	47.65	32.40	85.70	34.67	63.67
	SAGPool	44.00	**100.00**	24.31	29.85	25.88	37.65	46.65	56.15	42.40	81.70	33.44	**100.00**
GAT	TopkPool	52.00	**100.00**	31.08	84.62	32.15	88.23	30.15	88.65	47.30	96.00	59.22	**100.00**
	EdgePool	67.20	**100.00**	26.46	**99.08**	43.92	98.82	35.30	87.65	40.80	92.20	65.89	**100.00**
	ASAPool	52.80	88.80	18.15	88.31	35.68	77.65	25.25	**91.00**	14.70	75.20	48.11	94.33
	SAGPool	57.60	**100.00**	32.00	**100.00**	41.96	82.75	35.35	91.45	39.80	97.20	65.67	**100.00**
HO-GNN	TopkPool	54.40	**100.00**	24.92	70.46	23.53	90.59	23.90	75.30	22.60	97.80	53.67	**100.00**
	EdgePool	68.00	**100.00**	13.54	86.46	34.51	98.43	22.25	88.05	28.70	97.80	57.34	**100.00**
	ASAPool	52.00	69.60	13.85	61.54	29.80	75.69	20.25	85.90	22.60	82.30	34.33	77.56
	SAGPool	59.20	**100.00**	24.61	99.38	26.67	77.65	27.05	88.90	34.30	97.80	49.44	**100.00**

*TD means time domain input, and FD means frequency domain input.

Table A4
The results of graph-level fault diagnostics with PathGraph topology.

Model	Pooling Layer	CWRU		PU		MFPT		SEU		XJTUSpurgear		XJTUGearbox	
		TD	FD	TD	FD	TD	FD	TD	FD	TD	FD	TD	FD
SVM		84.00		87.69		74.51		88.50		74.50		96.11	
MLP	TopkPool	63.20	**100.00**	50.77	84.31	52.94	91.37	38.65	90.05	66.00	94.70	90.22	**100.00**
	EdgePool	**87.20**	**100.00**	43.08	83.69	57.65	99.22	24.55	87.95	45.80	**97.70**	89.78	**100.00**
	ASAPool	76.00	**100.00**	50.46	97.54	59.61	**100.00**	29.70	89.05	62.10	**99.80**	**93.00**	**100.00**
	SAGPool	61.60	96.00	36.00	81.85	49.02	89.02	21.65	88.65	64.90	96.50	86.22	**100.00**
ChebyNet	TopkPool	68.00	**100.00**	**74.46**	**97.54**	**61.57**	96.86	**62.05**	88.80	**76.90**	98.10	83.44	**100.00**
	EdgePool	85.60	**100.00**	**68.00**	**97.54**	**69.02**	98.82	**35.95**	**91.35**	**74.00**	95.20	**94.45**	**100.00**
	ASAPool	**80.00**	**100.00**	**65.54**	97.85	**67.06**	**100.00**	**57.20**	85.65	**75.80**	96.60	91.44	**100.00**
	SAGPool	**64.80**	99.20	**66.46**	96.31	61.57	**95.30**	**44.30**	86.75	**79.00**	99.40	89.44	**100.00**
GCN	TopkPool	54.40	**100.00**	59.38	91.38	49.80	**97.65**	34.95	90.20	64.80	92.90	88.78	**100.00**
	EdgePool	75.20	**100.00**	45.54	97.23	56.47	96.86	36.55	88.75	64.40	91.70	79.56	**100.00**
	ASAPool	66.40	**100.00**	51.69	**98.15**	59.61	96.86	47.90	91.35	68.80	**99.80**	85.11	**100.00**
	SAGPool	59.20	**100.00**	55.39	81.54	53.33	81.18	41.50	86.35	71.80	**99.00**	91.33	**100.00**
SGCN	TopkPool	63.20	**100.00**	47.38	90.46	52.94	71.37	23.75	86.10	41.50	94.40	83.33	**100.00**
	EdgePool	72.80	**100.00**	26.77	89.23	45.10	96.47	15.80	86.60	35.60	93.80	84.44	**100.00**
	ASAPool	67.20	**100.00**	44.31	91.08	49.02	95.69	26.05	87.40	46.10	95.40	69.89	**100.00**
	SAGPool	53.60	**100.00**	32.00	73.23	52.16	90.59	32.60	92.95	56.50	93.20	86.56	**100.00**
GraphSage	TopkPool	56.00	**100.00**	52.62	96.92	58.43	90.59	33.95	91.35	75.20	97.60	86.89	**100.00**
	EdgePool	84.80	**100.00**	37.23	94.15	39.22	90.98	29.80	88.05	59.20	95.50	82.55	**100.00**
	ASAPool	77.60	**100.00**	45.23	86.15	69.81	97.65	49.45	90.80	59.50	89.70	85.22	**100.00**
	SAGPool	59.20	**100.00**	49.85	**96.61**	**68.63**	92.55	32.20	90.60	60.40	86.40	93.56	**100.00**
GIN	TopkPool	57.60	**100.00**	19.39	53.54	30.20	74.51	15.05	73.15	26.60	80.50	49.78	**100.00**
	EdgePool	71.20	**100.00**	19.07	59.39	28.63	80.39	31.10	90.55	23.50	74.90	52.55	**100.00**
	ASAPool	58.40	**100.00**	16.92	40.31	23.14	65.49	28.45	83.75	21.40	86.00	44.11	**100.00**
	SAGPool	60.00	**100.00**	12.92	43.69	32.55	76.47	12.30	68.45	42.20	86.00	43.67	**100.00**
GAT	TopkPool	64.80	**100.00**	57.23	86.77	60.39	96.47	41.00	**92.00**	63.80	95.30	89.00	**100.00**
	EdgePool	77.60	**100.00**	50.46	94.15	56.08	**99.61**	35.05	86.65	58.40	97.50	**94.44**	**100.00**
	ASAPool	71.20	**100.00**	56.92	**98.15**	64.71	98.04	60.45	89.25	70.50	97.60	88.89	**100.00**
	SAGPool	61.60	**100.00**	60.92	89.54	68.23	82.35	**60**.75	**95.00**	65.70	95.70	**92.22**	**100.00**
HO-GNN	TopkPool	**69.60**	**100.00**	47.38	95.08	57.26	85.10	42.75	88.50	63.60	**98.90**	**93.67**	**100.00**
	EdgePool	80.00	**100.00**	26.46	83.08	60.78	98.04	31.90	88.70	54.70	95.80	87.00	**100.00**
	ASAPool	71.20	**100.00**	30.15	76.00	57.65	94.90	30.40	**92.15**	52.30	97.50	92.11	**100.00**
	SAGPool	64.00	**100.00**	36.62	73.54	43.92	83.53	24.95	92.60	61.10	93.00	90.44	**100.00**

*TD means time domain input, and FD means frequency domain input.

Table A5
The RUL prediction results of CMAPSS and PHM2010 datasets.

Model	Pooling Layer	CMAPSS								PHM2010					
		FD001		FD002		FD003		FD004		C_1		C_4		C_6	
		RMSE	SF	RMSE	SF	RMSE	SF	RMSE	SF	RMSE	SF	RMSE	SF	RMSE	SF
SVM		20.39	1820.24	28.42	8913.09	23.51	1535.88	34.37	57949.44	30.31	65311.91	45.31	548842.37	50.63	6004490.4
MLP	TopkPool	18.12	555.50	20.91	4897.56	16.04	554.78	**16.76**	**1509.39**	30.37	35402.54	41.33	130545.65	26.20	5578.19
	EdgePool	15.59	544.47	16.95	1827.33	14.28	451.31	17.47	2160.87	25.80	16607.63	18.70	2101.84	25.98	6986.15
	ASAPool	15.98	729.98	16.16	**1134.97**	16.65	671.47	**15.87**	1764.01	19.67	21280.85	18.08	1643.40	34.60	15205.11
	SAGPool	15.19	681.03	18.33	1945.61	15.48	589.63	19.15	3772.80	41.82	99642.32	34.11	19833.79	43.84	27752.95
ChebyNet	TopkPool	16.16	474.84	18.56	2016.17	**14.59**	497.61	17.77	2281.04	28.46	72457.52	25.41	6211.72	50.80	122293.53
	EdgePool	15.21	450.22	16.28	1229.22	14.67	421.02	**16.26**	**1159.74**	19.56	4738.18	**15.65**	**1281.35**	26.73	5178.65
	ASAPool	16.19	840.25	16.81	3274.64	15.70	480.51	16.45	**1501.50**	27.29	163967.09	29.84	35248.62	50.27	265045.13
	SAGPool	16.53	817.92	16.46	1470.33	15.82	590.49	19.61	5427.66	45.85	225500.81	50.24	870309.00	27.38	9540.28
GCN	TopkPool	15.37	1307.09	20.61	4612.17	14.61	**452.59**	21.00	3644.73	24.68	8304.30	17.47	2343.28	25.11	3355.40
	EdgePool	13.55	320.21	19.02	4317.00	13.82	381.87	20.41	2542.30	21.40	19804.25	24.57	6201.58	36.21	10723.07
	ASAPool	15.91	1379.80	22.71	8215.28	14.68	525.17	20.83	3148.83	39.39	336060.78	17.56	2190.75	27.92	4887.33
	SAGPool	15.34	1264.49	16.81	1850.82	14.99	507.14	19.52	2769.81	24.73	5893.99	22.94	10549.76	22.06	4815.76
SGCN	TopkPool	14.83	708.24	17.78	4146.09	15.47	492.61	24.27	10865.96	24.10	12754.22	24.06	70311.50	34.32	9045.24
	EdgePool	14.29	434.84	18.31	3009.14	14.17	387.84	20.13	2732.56	20.21	11649.52	18.05	4527.48	31.73	8651.72
	ASAPool	15.64	1172.11	17.88	2051.59	15.80	713.46	21.08	3561.73	20.53	5343.44	25.60	5484.19	30.11	6303.14
	SAGPool	14.71	373.98	18.61	2467.36	13.56	355.37	20.74	3467.38	22.28	8198.57	27.04	15952.47	25.70	3428.84
GraphSage	TopkPool	15.80	512.59	**15.62**	1620.86	16.08	532.61	18.52	2427.78	22.97	8205.67	34.89	21456.23	28.75	16290.47
	EdgePool	16.17	628.97	15.52	**1038.20**	14.64	642.33	17.49	1533.18	**18.55**	**1351.44**	20.76	3319.73	**21.15**	8466.26
	ASAPool	15.73	**469.19**	**15.68**	1164.39	14.43	495.71	17.60	1741.54	30.43	31468.92	15.38	2309.95	**23.22**	**2659.24**
	SAGPool	15.12	858.08	**16.00**	**1327.15**	**13.09**	391.18	**18.00**	**2154.96**	31.38	16680.04	23.90	6310.70	23.69	3999.97
GIN	TopkPool	16.20	514.33	16.86	**1506.77**	15.76	575.73	21.02	6928.37	26.30	8323.19	21.28	4360.40	25.86	3040.92
	EdgePool	15.84	472.92	18.30	2337.58	14.36	475.59	21.27	3283.47	23.58	4688.08	21.93	3755.30	27.83	5132.70
	ASAPool	16.58	651.21	18.09	2179.87	18.66	1411.91	21.12	3801.32	20.86	4258.64	22.08	5500.17	27.07	4133.34
	SAGPool	15.75	632.77	19.83	3233.39	14.48	600.57	21.50	4300.66	**19.34**	**2718.17**	**16.29**	**1305.17**	26.94	3725.07
GAT	TopkPool	**13.21**	**303.18**	17.25	5338.80	15.36	507.52	21.44	2971.93	19.87	3617.18	16.07	2108.98	29.01	224305.75
	EdgePool	**13.53**	**309.59**	**15.07**	1380.41	14.66	470.74	17.60	1726.53	25.12	76729.95	19.27	8078.60	32.61	9000.55
	ASAPool	**14.86**	983.16	18.06	2404.35	14.49	455.39	20.61	2443.29	22.88	173582.95	20.34	2867.57	26.91	4337.08
	SAGPool	**13.82**	**333.14**	18.52	3289.63	15.07	778.45	19.02	2262.72	23.75	117775.33	19.61	4793.27	25.60	3865.12
HO-GNN	TopkPool	17.20	955.96	17.09	3645.04	15.92	813.18	18.79	10586.32	26.82	12406.31	16.61	1556.87	27.52	5358.27
	EdgePool	16.13	678.58	16.16	1174.50	**13.10**	**329.97**	17.94	2327.09	24.83	7916.90	**14.94**	**984.18**	22.85	3833.35
	ASAPool	17.51	805.13	17.96	2160.46	**13.07**	**392.30**	19.16	2120.84	25.29	10778.40	19.56	3418.25	24.12	2890.74
	SAGPool	16.37	1219.45	17.17	2009.25	13.43	**360.21**	19.92	2908.82	26.50	11549.58	19.03	9879.74	31.06	6021.28

6.2. Embedding prior knowledge into the constructed graphs

The graph itself has certain prior knowledge [97], and the prior knowledge it contains will affect the final result, as shown in previous results. However, most works in fault diagnostics do not take into account prior knowledge when constructing graphs, and the commonly used methods for graph construction are the aforementioned three types of methods. How to embed the prior knowledge into the process of graph construction could be a future research direction.

6.3. Generalize to existing models

GCN as an emerging architecture can be generalized to existing DNN architectures. For example, we can intuitively replace the layers in AEs and RNNs by GConv layers to get GAEs and GRNNs. Apart from extending GConv layers to existing models, we can also construct a model to incorporate the structural information of the graph into the model. For example, Graphormer [98] proposed three kinds of encoding mechanisms, including centrality encoding, spatial encoding, and edge encoding, to embed the graph structure information into the extracted features, and through such three encoding mechanisms, Graphormer obtained a very promising performance for graph representation. In addition to designing new building blocks, how to effectively and systematically integrate these architectures is a meaningful future direction. During the process, how to design the architecture and incorporate domain knowledge is also an open problem. Besides, AutoML is also a good way to assemble different components and select hyperparameters [99,100].

6.4. Interpretability of GNNs

Since graphs are usually related to risk-sensitive scenarios, the ability to interpret the results of GNNs is crucial in decision-making. However, since the nodes and edges of a graph are usually heavily interconnected, the interpretability of GNNs is more challenging than other black box models. How to make the results of GNNs interpretable requires further research. For example, graph wavelet neural network (GWNN) [101], embeds the concept of graph wavelet transform in GCN, which makes GWNN more efficient and has good interpretability.

7. Conclusions

In this paper, a practical guideline is proposed and a novel intelligent fault diagnostics and prognostics framework based on GNN is established. In this framework, three types of graph construction methods are provided, and seven GCNs with four graph pooling methods are investigated. The benchmark study is also carried out on eight datasets, including six fault diagnosis datasets and two prognosis datasets. Based on the benchmark results, it is found that the overall performance of ChebyNet on these eight datasets is best, and ChebyNet with EdegPool can also achieve the best results in the two kinds of graph-level tasks. In the discussion part, we find that the reason why GCNs work is that the graph convolution operation enables the features of the nodes on the graph to be similar, thus facilitating node-level fault classification. Besides, it is also proved that residual connection does help to design a deeper GCN. Moreover, the influence of aggregators on model performance and the robustness of GCNs are also explored. Finally, four potential further directions are provided.

CRediT authorship contribution statement

Tianfu Li: Conceptualization, Methodology, Software, Data curation, Writing – original draft. **Zheng Zhou:** Data curation, Writing – review & editing. **Sinan Li:** Data curation, Writing – review & editing. **Chuang Sun:** Supervision, Conceptualization, Data curation, Writing – review & editing. **Ruqiang Yan:** Supervision. **Xuefeng Chen:** Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Mr. Yang Yuangui for his help in revising the paper.

This work was supported in part by Natural Science Foundation of China (No. 52175116) and Major Research Program of Natural Science Foundation of China (No. 92060302), Shaanxi province 2020 natural science basic research plan (No. 2020JQ-042), National Key Science and Technology Infrastructure Opening Project Fund for Research and Evaluation facilities for Service Safety of Major Engineering Materials, Aeronautical Science Foundation(No. 2019ZB070001, No. 20200046070002), National Science and Technology Major Project (J2019-IV-0018-0086), and the Fundamental Research Funds for the Central Universities.

Appendix

References

[1] H. Ding, R.X. Gao, A.J. Isaksson, R.G. Landers, T. Parisini, Y.e. Yuan, State of AI-based monitoring in smart manufacturing and introduction to focused section, IEEE/ASME Trans. Mechatron. 25 (5) (2020) 2143–2154.
[2] Z. Gao, C. Cecati, S.X. Ding, A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches, IEEE Trans. Ind. Electron. 62 (6) (2015) 3757–3767.
[3] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, A.K. Nandi, Applications of machine learning to machine fault diagnosis: A review and roadmap, Mech. Syst. Sig. Process. 138 (2020) 106587, https://doi.org/10.1016/j.ymssp.2019.106587.
[4] T. Li, Z. Zhao, C. Sun, R. Yan, X. Chen, Adaptive channel weighted CNN with multisensor fusion for condition monitoring of helicopter transmission system, IEEE Sens. J. 20 (15) (2020) 8364–8373.
[5] T. Wang, Q. Han, F. Chu, Z. Feng, Vibration based condition monitoring and fault diagnosis of wind turbine planetary gearbox: A review, Mech. Syst. Sig. Process. 126 (2019) 662–685.
[6] P. Mercorelli, P. Terwiesch, A black box identification in frequency domain, Eur. Trans. Electr. Power 13 (1) (2003) 29–40.
[7] P. Mercorelli, Denoising and harmonic detection using nonorthogonal wavelet packets in industrial applications, J. Syst. Sci. Complexity 20 (3) (2007) 325–343.
[8] P. Mercorelli, Biorthogonal wavelet trees in the classification of embedded signal classes for intelligent sensors using machine learning applications, J. Franklin Inst. 344 (6) (2007) 813–829.
[9] M. Nentwig, P. Mercorelli. Throttle valve control using an inverse local linear model tree based on a fuzzy neural network, proceedings of the 2008 7th IEEE International Conference on Cybernetic Intelligent Systems. 2008. IEEE,1-6.
[10] Z. Chen, K. Gryllias, W. Li, Mechanical fault diagnosis using convolutional neural networks and extreme learning machine, Mech. Syst. Sig. Process. 133 (2019) 106272, https://doi.org/10.1016/j.ymssp.2019.106272.
[11] F. Jia, Y. Lei, J. Lin, X. Zhou, N.a. Lu, Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data, Mech. Syst. Sig. Process. 72-73 (2016) 303–315.
[12] Z.-b. Yang, J.-P. Zhang, Z.-B. Zhao, Z. Zhai, X.-F. Chen, Interpreting network knowledge with attention mechanism for bearing fault diagnosis, Appl. Soft Comput. 97 (2020) 106829, https://doi.org/10.1016/j.asoc.2020.106829.
[13] Z. Chen, A. Mauricio, W. Li, K. Gryllias, A deep learning method for bearing fault diagnosis based on cyclic spectral coherence and convolutional neural networks, Mech. Syst. Sig. Process. 140 (2020) 106683, https://doi.org/10.1016/j.ymssp.2020.106683.
[14] T. Li, Z. Zhao, C. Sun, et al., WaveletKernelNet: An interpretable deep neural network for industrial intelligent diagnosis. IEEE Transactions on Systems, Man, and Cybernetics: Systems, IEEE, 2021, https://doi.org/10.1109/TSMC.2020.3048950.
[15] H. Shao, H. Jiang, H. Zhang, T. Liang, Electric locomotive bearing fault diagnosis using a novel convolutional deep belief network, IEEE Trans. Ind. Electron. 65 (3) (2018) 2727–2736.
[16] Z. He, H. Shao, P. Wang, J.(. Lin, J. Cheng, Y.u. Yang, Deep transfer multi-wavelet auto-encoder for intelligent fault diagnosis of gearbox with few target training samples, Knowl.-Based Syst. 191 (2020) 105313, https://doi.org/10.1016/j.knosys.2019.105313.
[17] F. Xu, Z. Huang, F. Yang, D. Wang, K.L. Tsui, Constructing a health indicator for roller bearings by using a stacked auto-encoder with an exponential function to eliminate concussion, Appl. Soft Comput. 89 (2020) 106119, https://doi.org/10.1016/j.asoc.2020.106119.
[18] H. Zhu, J. Cheng, C. Zhang, J. Wu, X. Shao, Stacked pruning sparse denoising autoencoder based intelligent fault diagnosis of rolling bearings, Appl. Soft Comput. 88 (2020) 106060, https://doi.org/10.1016/j.asoc.2019.106060.
[19] M. Ma, C. Sun, X. Chen, Discriminative deep belief networks with ant colony optimization for health status assessment of machine, IEEE Trans. Instrum. Meas. 66 (12) (2017) 3115–3125.
[20] Y. Wang, Z. Pan, X. Yuan, C. Yang, W. Gui, A novel deep learning based fault diagnosis approach for chemical process with extended deep belief network, ISA Trans. 96 (2020) 457–467.
[21] X. Yan, Y. Liu, M. Jia, Multiscale cascading deep belief network for fault identification of rotating machinery under various working conditions, Knowl.-Based Syst. 193 (2020) 105484, https://doi.org/10.1016/j.knosys.2020.105484.
[22] Z. An, S. Li, J. Wang, X. Jiang, A novel bearing intelligent fault diagnosis framework under time-varying working conditions using recurrent neural network, ISA Trans. 100 (2020) 155–170.
[23] X. Ji, Y. Ren, H. Tang, J. Xiang, DSmT-based three-layer method using multi-classifier to detect faults in hydraulic systems, Mech. Syst. Sig. Process. 153 (2021) 107513, https://doi.org/10.1016/j.ymssp.2020.107513.
[24] R.-B. Sun, Z.-B. Yang, L.-D. Yang, B.-J. Qiao, X.-F. Chen, K. Gryllias, Planetary gearbox spectral modeling based on the hybrid method of dynamics and LSTM, Mech. Syst. Sig. Process. 138 (2020) 106611, https://doi.org/10.1016/j.ymssp.2019.106611.
[25] J. Zhu, N. Chen, C. Shen, A new data-driven transferable remaining useful life prediction approach for bearing under different working conditions, Mech. Syst. Sig. Process. 139 (2020) 106602, https://doi.org/10.1016/j.ymssp.2019.106602.
[26] K. Yu, T.R. Lin, H. Ma, X. Li, X.u. Li, A multi-stage semi-supervised learning approach for intelligent fault diagnosis of rolling bearing using data augmentation and metric learning, Mech. Syst. Sig. Process. 146 (2021) 107043, https://doi.org/10.1016/j.ymssp.2020.107043.
[27] S. Haidong, J. Hongkai, Z. Ke, W. Dongdong, L.i. Xingqiu, A novel tracking deep wavelet auto-encoder method for intelligent fault diagnosis of electric locomotive bearings, Mech. Syst. Sig. Process. 110 (2018) 193–209.
[28] J. Jiao, M. Zhao, J. Lin, K. Liang, A comprehensive review on convolutional neural network in machine fault diagnosis, Neurocomputing. 417 (2020) 36–63.
[29] K. Chen, J. Hu, Y.u. Zhang, Z. Yu, J. He, Fault location in power distribution systems via deep graph convolutional networks, IEEE J. Sel. Areas Commun. 38 (1) (2020) 119–131.
[30] T. Li, Z. Zhao, C. Sun, et al., Multireceptive Field Graph Convolutional Networks for Machine Fault Diagnosis, IEEE Trans. Ind. Electron. 68 (12) (2021) 12739–12749, https://doi.org/10.1109/TIE.2020.3040669.
[31] X. Zhao, M. Jia, Z. Liu, Semi-supervised graph convolution deep belief network for fault diagnosis of electormechanical system with limited labeled data, IEEE Trans. Ind. Inf. 17 (8) (2021) 5450–5460.
[32] T. Li, Z. Zhao, C. Sun, R. Yan, X. Chen, Domain Adversarial Graph Convolutional Network for Fault Diagnosis Under Variable Working Conditions, IEEE Trans. Instrum. Meas. 70 (2021) 1–10.
[33] Z. Zhang, P. Cui, W. Zhu, Deep learning on graphs: A survey, IEEE Trans. Knowl. Data Eng. (2020).
[34] M. Chen, Z. Wei, Z. Huang, et al. Simple and deep graph convolutional networks,proceedings of the International Conference on Machine Learning. 2020. PMLR,1725-35.
[35] L. Ruiz, F. Gama, A. Ribeiro, Gated graph recurrent neural networks, IEEE Trans. Signal Process. 68 (2020) 6303–6318.
[36] V. Rennard, G. Nikolentzos, M. Vazirgiannis. Graph Auto-Encoders for Learning Edge Representations,proceedings of the International Conference on Complex Networks and Their Applications. 2020. Springer,117-29.
[37] Z. Wu, S. Pan, F. Chen, et al., A comprehensive survey on graph neural networks, IEEE transactions on neural networks and learning systems. (2020).
[38] S. Ye, J. Liang, R. Liu, X.i. Zhu, Symmetrical Graph Neural Network for Quantum Chemistry with Dual Real and Momenta Space, The Journal of Physical Chemistry A. 124 (34) (2020) 6945–6953.
[39] X. Chen, S. Jia, Y. Xiang, A review: Knowledge reasoning over knowledge graph, Expert Syst. Appl. 141 (2020) 112948, https://doi.org/10.1016/j.eswa.2019.112948.
[40] L. Yao, C. Mao, Y. Luo. Graph convolutional networks for text classification,proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence. 2019.7370-7.
[41] L. Wu, P. Sun, R. Hong, et al., SocialGCN: an efficient graph convolutional network based model for social recommendation, arXiv preprint arXiv:181102815. (2018).

[42] H. Peng, H. Wang, B. Du, M.Z.A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang, P.S. Yu, Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting, Inf. Sci. 521 (2020) 277–290.
[43] D. Bacciu, F. Errica, A. Micheli, M. Podda, A gentle introduction to deep learning for graphs, Neural Networks 129 (2020) 203–221.
[44] Y. Gao, M. Chen, D. Yu, Semi-supervised graph convolutional network and its application in intelligent fault diagnosis of rotating machinery, Measurement 186 (2021) 110084, https://doi.org/10.1016/j.measurement.2021.110084.
[45] C. Li, L. Mo, R. Yan, Fault Diagnosis of Rolling Bearing Based on WHVG and GCN, IEEE Trans. Instrum. Meas. 70 (2021) 1–11.
[46] B. Zhao, X. Zhang, Z. Zhan, et al., Multi-scale Graph-guided Convolutional Network with Node Attention for Intelligent Health State Diagnosis of a 3-PRR Planar Parallel Manipulator, IEEE Transactions on Industrial Electronics. (2021).
[47] D. Zhang, E. Stewart, M. Entezami, C. Roberts, D. Yu, Intelligent acoustic-based fault diagnosis of roller bearings using a deep graph convolutional network, Measurement 156 (2020) 107585, https://doi.org/10.1016/j.measurement.2020.107585.
[48] X. Yu, B. Tang, K. Zhang, Fault Diagnosis of Wind Turbine Gearbox Using a Novel Method of Fast Deep Graph Convolutional Networks, IEEE Trans. Instrum. Meas. 70 (2021) 1–14.
[49] C. Li, L. Mo, R. Yan. Rolling Bearing Fault Diagnosis Based on Horizontal Visibility Graph and Graph Neural Networks,proceedings of the 2020 International Conference on Sensing, Measurement & Data Analytics in the era of Artificial Intelligence (ICSMD). 2020. IEEE,275-9.
[50] X. Xu, Z. Tao, W. Ming, Q. An, M. Chen, Intelligent monitoring and diagnostics using a novel integrated model based on deep learning and multi-sensor feature fusion, Measurement 165 (2020) 108086, https://doi.org/10.1016/j.measurement.2020.108086.
[51] L. Jing, T. Wang, M. Zhao, P. Wang, An adaptive multi-sensor data fusion method based on deep convolutional neural networks for fault diagnosis of planetary gearbox, Sensors. 17 (2) (2017) 414, https://doi.org/10.3390/s17020414.
[52] A. Al-Dulaimi, S. Zabihi, A. Asif, A. Mohammadi, A multimodal and hybrid deep neural network model for remaining useful life estimation, Comput. Ind. 108 (2019) 186–196.
[53] P. Shan, H. Lv, L. Yu, H. Ge, Y. Li, L.e. Gu, A multisensor data fusion method for ball screw fault diagnosis based on convolutional neural network with selected channels, IEEE Sens. J. 20 (14) (2020) 7896–7905.
[54] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks 20 (1) (2009) 61–80.
[55] C. Gallicchio, A. Micheli. Graph echo state networks,proceedings of the The 2010 International Joint Conference on Neural Networks (IJCNN). 2010. IEEE,1-8.
[56] Y. Li, D. Tarlow, M. Brockschmidt, et al., Gated graph sequence neural networks, arXiv preprint arXiv:151105493. (2015).
[57] H. Dai, Z. Kozareva, B. Dai, et al. Learning steady-states of iterative algorithms over graphs,proceedings of the International conference on machine learning. 2018. PMLR,1106-14.
[58] Z. Wu, S. Pan, G. Long, et al., Graph wavenet for deep spatial-temporal graph modeling, arXiv preprint arXiv:190600121. (2019).
[59] J. Bruna, W. Zaremba, A. Szlam, et al., Spectral networks and locally connected networks on graphs, arXiv preprint arXiv:13126203. (2013).
[60] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv preprint arXiv:150605163. (2015).
[61] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, arXiv preprint arXiv:160609375. (2016).
[62] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:160902907. (2016).
[63] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Trans. Neural Networks 20 (3) (2009) 498–511.
[64] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, arXiv preprint arXiv:170602216. (2017).
[65] P. Veličković, G. Cucurull, A. Casanova, et al., Graph attention networks, arXiv preprint arXiv:171010903. (2017).
[66] K. Xu, W. Hu, J. Leskovec, et al., How powerful are graph neural networks? , arXiv preprint arXiv:181000826. (2018).
[67] J. Zhou, G. Cui, Z. Zhang, et al., Graph neural networks: A review of methods and applications, arXiv preprint arXiv:181208434. (2018).
[68] D.I. Shuman, S.K. Narang, P. Frossard, A. Ortega, P. Vandergheynst, The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains, IEEE Signal Process Mag. 30 (3) (2013) 83–98.
[69] F. Wu, A. Souza, T. Zhang, et al. Simplifying graph convolutional networks,proceedings of the International conference on machine learning. 2019. PMLR,6861-71.
[70] C. Morris, M. Ritzert, M. Fey, et al. Weisfeiler and leman go neural: Higher-order graph neural networks,proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence. 2019.4602-9.
[71] C. Cangea, P. Veličković, N. Jovanović, et al., Towards sparse hierarchical graph classifiers, arXiv preprint arXiv:181101287. (2018).
[72] J. Lee, I. Lee, J. Kang. Self-attention graph pooling,proceedings of the International Conference on Machine Learning. 2019. PMLR,3734-43.
[73] F. Diehl, Edge contraction pooling for graph neural networks, arXiv preprint arXiv:190510990. (2019).
[74] E. Ranjan, S. Sanyal, P. Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations,proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence. 2020.5470-7.
[75] A. Paszke, S. Gross, F. Massa, et al., Pytorch: An imperative style, high-performance deep learning library, arXiv preprint arXiv:191201703. (2019).
[76] M. Fey, J. E. Lenssen, Fast graph representation learning with PyTorch Geometric, arXiv preprint arXiv:190302428. (2019).
[77] P. Park, P.D. Marco, H. Shin, J. Bang, Fault detection and diagnosis using combined autoencoder and long short-term memory network, Sensors. 19 (21) (2019) 4612, https://doi.org/10.3390/s19214612.
[78] A. Saxena, K. Goebel, D. Simon, et al. Damage propagation modeling for aircraft engine run-to-failure simulation,proceedings of the 2008 international conference on prognostics and health management. 2008. IEEE,1-9.
[79] J. Zhang, P. Wang, R. Yan, R.X. Gao, Long short-term memory for machine remaining life prediction, J. Manuf. Syst. 48 (2018) 78–86.
[80] M. Naumov, L. Chien, P. Vandermersch, et al., Cusparse library, proceedings of the GPU, Technology Conference. (2010).
[81] Case Western Reserve University Bearing Data Center. [online]. http://csegroups.case.edu/bearingdatacenter/pages/welcomecase-western-reserve-university-bearing-data-center-website.
[82] C. Lessmeier, J. K. Kimotho, D. Zimmer, et al., Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: A benchmark data set for data-driven classification, Proceedings of the European conference of the prognostics and health management society. (2016), 05-8.
[83] D. Lee, V. Siu, R. Cruz, et al., Convolutional neural net and bearing fault analysis, Proceedings of the International Conference on Data Science (ICDATA). (2016), 194.
[84] S. Shao, S. McAleer, R. Yan, P. Baldi, Highly accurate machine fault diagnosis using deep transfer learning, IEEE Trans. Ind. Inf. 15 (4) (2019) 2446–2455.
[85] A. Saxena, K. Goebel, Turbofan engine degradation simulation data set, NASA Ames Prognostics Data Repository. (2008), 1551-3203.
[86] The Prognostics and Health Management Society (PHM Society), [online]. https://www.phmsociety.org/competition/phm/10.
[87] T. Li, Z. Zhao, C. Sun, R. Yan, X. Chen, Hierarchical attention graph convolutional network to fuse multi-sensor signals for remaining useful life prediction, Reliab. Eng. Syst. Saf. 215 (2021) 107878, https://doi.org/10.1016/j.ress.2021.107878.
[88] Z. Zhao, T. Li, J. Wu, C. Sun, S. Wang, R. Yan, X. Chen, Deep learning algorithms for rotating machinery intelligent diagnosis: An open source benchmark study, ISA Trans. 107 (2020) 224–255.
[89] X. Li, Q. Ding, J.-Q. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks, Reliab. Eng. Syst. Saf. 172 (2018) 1–11.
[90] F. Chen, B. Tang, T. Song, L.i. Li, Multi-fault diagnosis study on roller bearing based on multi-kernel support vector machine with chaotic particle swarm optimization, Measurement 47 (2014) 576–590.
[91] L.-l. Jiang, H.-K. Yin, X.-J. Li, S.-W. Tang, Fault diagnosis of rotating machinery based on multisensor information fusion using SVM and time-domain features, Shock Vib. 2014 (2014) 1–8.
[92] Q. Li, Z. Han, X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning, proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence. 2018.
[93] G. Li, C. Xiong, A. Thabet, et al., Deepergcn: All you need to train deeper gcns, arXiv preprint arXiv:200607739. (2020).

[94] G. Li, M. Muller, A. Thabet, et al. Deepgcns: Can gcns go as deep as cnns?, proceedings of the Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.9267-76.
[95] Y. Shi, A. Deng, M. Deng, J. Zhu, Y. Liu, Q. Cheng, Enhanced lightweight multiscale convolutional neural network for rolling bearing fault diagnosis, IEEE Access 8 (2020) 217723–217734.
[96] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond euclidean data, IEEE Signal Process Mag. 34 (4) (2017) 18–42.
[97] Z. Chen, J. Xu, T. Peng, et al., Graph Convolutional Network-Based Method for Fault Diagnosis Using a Hybrid of Measurement and Prior Knowledge, IEEE Transactions on Cybernetics. (2021).
[98] C. Ying, T. Cai, S. Luo, et al., Do Transformers Really Perform Bad for Graph Representation? , arXiv preprint arXiv:210605234. (2021).
[99] Y. Gao, H. Yang, P. Zhang, et al., Graphnas: Graph neural architecture search with reinforcement learning, arXiv preprint arXiv:190409981. (2019).
[100] Z. Zhou, T. Li, Z. Zhang, et al., Bayesian differentiable architecture search for efficient domain matching fault diagnosis, IEEE Trans. Instrum. Meas. 70 (2021) 1–11.
[101] B. Xu, H. Shen, Q. Cao, et al., Graph wavelet neural network, arXiv preprint arXiv:190407785. (2019).

Tianfu Li, received the B.S. degree in Mechanical Engineering from Chongqing University, China in 2018. He is currently working towards the Ph.D. degree in mechanical engineering in the Department of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, China. His current research is focused on deep learning, mechanical fault diagnosis and prognosis.

Zheng Zhou, received the B.S. degree in mechanical engineering in the Department of Mechanical Engineering, Xi'an Jiaotong University, China, in 2019. He is currently working towards the Ph.D. degree in mechanical engineering in the Department of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, China. His current research is focused on physical-informed machine learning, automatic machine learning, and fault prognosis.

Sinan Li, received the B.S. degree in Mechanical Engineering from Xi'an Jiaotong University, China in 2020. He is currently working towards the Ph.D. degree in mechanical engineering in the Department of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, China. His current research is focused on explainable deep learning, mechanical fault diagnosis and prognosis.

Chuang Sun, received the Ph.D. degree in mechanical engineering from Xi'an Jiaotong University, Xi'an, China, in 2014. From Mar. 2015 to Mar. 2016, he was a postdoc at Case Western Reserve University, USA. He is now an Associate Professor in School of Mechanical Engineering at Xi'an Jiaotong University. His research is focused on manifold learning, deep learning, sparse representation, mechanical fault diagnosis and prognosis, remaining useful life prediction.

Ruqiang Yan, (M'07, SM'11) received the M.S. degree in precision instrument and machinery from the University of Science and Technology of China, Hefei, China, in 2002, and the Ph.D. degree in mechanical engineering from the University of Massachusetts at Amherst, Amherst, MA, USA, in 2007. From 2009 to 2018, he was a Professor with the School of Instrument Science and Engineering, Southeast University, Nanjing, China. He joined the School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, China, in 2018. His reearch interests include data analytics, machine learning, and energy-efficient sensing and sensor networks for the condition monitoring and health diagnosis of large-scale, complex, dynamical systems. Dr. Yan is a Fellow of ASME (2019) and IEEE (2021). His honors and awards include the IEEE Instrumentation and Measurement Society Technical Award in 2019, the New Century Excellent Talents in University Award from the Ministry of Education in China in 2009, and multiple best paper awards. He is also the Editor-in-Chief of the IEEE Transactions on Instrumentation and Measurement and an Associate Editor of the IEEE Systems Journal and the IEEE Sensors Journal.

Xuefeng Chen, (M'12) received the Ph.D. degree from Xi'an Jiaotong University, Xi'an, China, in 2004. He is currently a Professor of Mechanical Engineering with Xi'an Jiaotong University. His current research interests include finite-element method, mechanical system and signal processing, diagnosis and prognosis for complicated industrial systems, smart structures, aero-engine fault diagnosis, and wind turbine system monitoring. Dr. Chen was a recipient of the National Excellent Doctoral Dissertation of China in 2007, the Second Award of Technology Invention of China in 2009, the National Science Fund for Distinguished Young Scholars in 2012, and a Chief Scientist of the National Key Basic Research Program of China (973 Program) in 2015. He is the Chapter Chairman of the IEEE Xi'an and Chengdu Joint Section Instrumentation and Measurement Society.