A Class-Imbalanced Heterogeneous Federated Learning Model for Detecting Icing on Wind Turbine Blades

Xu Cheng, *Member, IEEE,* Fan Shi, Yongping Liu, Jiehan Zhou, *Member, IEEE,* Xiufeng Liu, Lizhen Huang

***Abstract*—Wind farms are typically located at high latitudes, resulting in a high risk of blade icing. Data-driven approaches offer promising solutions for blade icing detection, but they rely on a considerable amount of data. Data exchange between multiple wind farms would improve the performance of detection models, due to the spatio-temporal dependencies capable of reflecting different meteorological conditions. The traditional centralized approach for icing detection faces many challenges, including the requirement of high storage and computational capacity of the server, vulnerability to cyberattacks, and operators' reluctance of sharing data for commercial reasons. To address these challenges, this paper proposes a heterogeneous federated learning (FL) model for wind turbine blade icing detection. The structures of the server and client models in the presented method are different, in contrast to the traditional FL of sharing the same structure. In addition, this paper addresses the class imbalance problem in the training data. Last, this paper conducts comprehensive experiments to evaluate the proposed method using real-world data from 20 turbines in two wind farms, and compares it with two state-of-the-art FL models and five well-known class imbalance methods. The experimental results verify the effectiveness and superiority of the proposed method.**

***Index Terms*—Wind Turbine, Blade Icing Detection, Federated Learning, Imbalanced Learning, Heterogeneous Structure**

I. Introduction

WIND energy technology has developed rapidly, alongside the more mature technology of wind turbines. However, blade icing considerably limits system performance, especially when turbines are installed in high-latitude areas. In severe cases, blade icing can reduce worldwide annual power generation by nearly 30% [1].

Traditionally, human observations, passive and active methods have been the three main means of detecting blade icing. Human observation relies heavily on observers' expertise. Passive methods use special materials such as black paint and coating for anti-icing purposes [2], but prevention through coating alone is not realistic. Active methods require additional power and mechanical replacement, which can damage the turbines [1]. To complement these traditional methods of blade icing detection, mathematical and data-driven approaches are receiving increased attention. Mathematical methods predict blade icing using mathematical or numerical models [3]. However, the disadvantages are obvious. First, they depend heavily on certain assumptions that can misidentify icing conditions. Second, they require external experimental tools such as wind tunnels to create accurate models. Finally, they often require domain knowledge to understand the icing process [4]. In recent years, with the increased use of Internet of Things technologies for monitoring purposes, a large amount of data has been generated that can be used to build data-driven models for detecting blade icing. A key advantage is that such models are not dependent on prior domain knowledge, but rather on available data, which can be more efficient and cost-effective [5]. Thus, such models are becoming increasingly popular in academia and industry, and in particular, end-to-end solutions based on deep neural networks have attracted significant research interest.

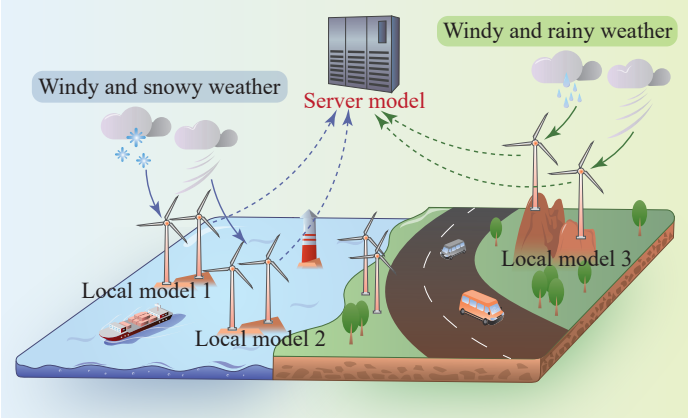

Fig. 1. Illustration of the proposed BiFL for blade-icing detection under different weather conditions

Data-driven models, such as deep neural networks, require a large amount of data and high computing capacity for training. Data from different wind farms help to improve the performance of the detection model, due to the spatio-temporal dependencies reflecting different weather conditions at different locations [6]. The traditional approach usually collects data from different sources to a central server and

Corresponding author: Xiufeng Liu, Lizhen Huang, (e-mail:xiuli@dtu.dk, lizhen.huang@ntnu.no.). Xu Cheng and Fan Shi are equal contribution.

Xu Cheng, Yongping Liu, and Lizhen Huang are with the Department of Manufacturing and Civil Engineering, Norwegian University of Science and Technology, Gjøvik, 2815, Norway e-mail: xu.cheng@ieee.org, yongping.liu@ntnu.no, lizhen.huang@ntnu.no.

Xiufeng Liu is with Department of Technology, Management and Economics, Technical University of Denmark, Produktionstorvet, 2800, Denmark e-mail: xiuli@dtu.dk.

Fan Shi is with the Key Laboratory of Computer Vision and System of Ministry of Education, School of Computer Science and Technology, Tianjin University of Technology, Tianjin, 300384, China.

Jiehan Zhou is with the Network Information Systems, Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, TS352, Finland.

then trains the model. This has several drawbacks. First, it requires intensive computing resources, including large data storage space and high computing capacity; second, transferring data from source systems to the central server requires large network bandwidth and additional communication cost; third, the data in the network communication and in the central storage are vulnerable to cyber-attacks. Most importantly, most wind farms are unwilling to share their data, mainly due to commercial competition. The sensor data of wind farms often contain valuable information that helps maintain a competitive position in the market, such as the key parameters for tuning and controlling wind turbines for better health and improved performance. This situation has resulted in the so-called "data island" problem. To address the above challenges, it has become necessary to build a distributed learning model to maximize the value of the data and improve collaboration among farms. Federated learning (FL) could be an ideal solution [7]. An FL approach obtains a global model by combining multiple local models trained distributively at different physical locations, meaning that no raw data have to be transferred to a central server or shared with others.

Nevertheless, applying a conventional FL framework to blade icing detection has the following limitations. First, an FL model usually has the same neural network architecture on both the server and client sides. In this study, wind turbines are located in remote areas, either onshore or offshore, and have very limited computing resources, making it difficult to train a neural network model. If a shared small neural network is applied to each client and the server, the learning ability of the FL model will be compromised, and the blade icing detection capabilities will be impaired. If a larger model is applied to each client and server, this would be constrained by the computing resources of clients. Therefore, applying the same neural network architecture to both server and clients is impractical. Second, the sensor data collected from each client is class-imbalanced. In other words, the class distribution is highly skewed between icing and non-icing (i.e., normal) samples, as wind turbines usually operate most of the time. In addition, sensor data are not independent and identically distributed, which can significantly affect the performance of the common model. Third, blade icing is closely related to the specification of wind turbines. For example, if there are two turbines with different sizes (e.g., one is 30kWh while the other is 1MW) at the same location under slight icing conditions, the small-sized turbine will be iced, while the bigger one may remain clean. Therefore, models should be constructed specifically for different types of wind turbines. In this case, the traditional FL model, where the server and all clients share the same architecture, becomes no longer applicable.

To address the above challenges, we propose a novel heterogeneous FL model for collaborative blade icing detection, called *BiFL (blade icing federated learning)*, which can be applied to different types of wind turbines under various weather conditions, as illustrated in Fig. 1. Compared to conventional FL, the model structures between the clients and the server in BiFL are allowed to be different. Clients have a small deep neural network model, while the server has a large model to fit all models to their available computing resources. Unlike the traditional FL model, where model gradients are transferred between clients and the server, the proposed BiFL only transfers the hidden features learned. To address the data imbalance problem, we introduce a novel class imbalance method using prototypes. Based on the extracted features, a prototype is first learned for each class, and then a classifier is built from it. In addition, BiFL incorporates the concept of transfer learning implemented by knowledge distillation (KD) to enhance knowledge transfer between the clients and the server and the robustness and generalization of the detection model.

In summary, the contributions of this paper are three-fold:

1) We propose BiFL, an FL-based method for blade-icing detection of wind turbines. This is the first work to leverage FL for blade-icing detection to the best of our knowledge. BiFL uses a heterogeneous structure between the clients and the server to learn a global model, which allows generating a model by maximizing the use of data from different data owners.
2) We propose a new method to solve the class imbalance problem by a prototype-based approach. Instead of sampling the raw data or tuning the learning algorithm, it balances the features extracted by a neural network. We also propose an information protection mechanism to avoid data breaches without compromising the performance of the model.
3) We comprehensively evaluated BiFL by comparing it with two state-of-the-art FL models and five well-known class imbalance methods. The results demonstrate the superiority of our model in blade icing detection. An ablation study validated the effectiveness of each component of BiFL, and a sensitivity study evaluated the influence of key parameters.

The rest of the study is structured as follows. Section II reviews the literature of wind turbine blade icing detection and FL. Section III describes the proposed BiFL model. Section IV conducts the evaluation. Section V concludes the paper and presents the future research directions.

II. Related Work

A. Blade icing detection

The literature has proposed passive and active de-icing systems and model- and data-driven methods for blade icing detection in wind turbines. Passive methods use special materials such as liquid-infused surfaces [8] and porous superhydrophobic/polyvinylidene fluoride coatings [9], while active methods employ external sensors or tools [10]. In model-driven methods, mathematical model draws the relationship between weather-related parameters (such as temperature, humidity, and wind speed) and blade icing according to physical characteristics of blade icing [11], [12]. There are two typical mathematical models for blade icing: the empirical statistical model (ESM) and ice growth model (IGM). ESM is established based on long-term monitoring data of a specific area. And the frequency and quantity of ice can be estimated based on the analysis of a large number of historical data. It

is obvious to know that the ESM is not accurate. IGM regards blade icing as a complex which combines aerodynamics, gas-liquid two-phase hydrodynamics, and heat transfer. IGM is good to understand the icing process, but it highly depends on the domain knowledge and external experiment tools, such as wind tunnels [4]. In recent years, data-driven methods have received increasing attention, especially with the emergence of deep learning, which has the robust ability to extract features not based on prior knowledge. Jiménez et al. used machine learning algorithms, including decision trees and support vector machines, to implement a classifier to identify the presence and thickness of blade ice on wind turbines from ultrasonic signals [13]. Liu et al. proposed an ensemble depth-learning model to extract multilevel features directly from SCADA data [14]. Yuan et al. presented a wavelet-based CNN model [15] that can achieve competitive performance over traditional machine learning models. Cheng et al. introduced a novel temporal CNN for blade-icing detection [5]. In all of the above studies, data were collected from geographically distributed wind farms, stored in a central location (such as the Cloud), and then used to create blade icing detection models. However, this centralized modeling approach requires not only large computational sources but also a large storage facility, which can be prohibitively expensive in many cases. In addition, transferring data over the network may lead to communication overhead and is vulnerable to cyberattack. For business reasons, wind farms may not wish to share their data [6]. This paper addresses these challenges by proposing an FL-based framework for wind turbine blade icing detection, which can fully utilize the wind farm SCADA data at different geographical locations. With the FL framework, the model is trained distributively across the physical locations of wind farms. To the best of our knowledge, this is the first attempt to address the problem of blade icing detection with the FL framework.

B. Federated learning

In recent years, data protection has received unprecedented attention in the field of energy. As a result, researchers begin exploring more efficient and effective learning approaches, among which FL has proven to be a promising one [16]. Zhang et al. proposed a novel federated probabilistic forecasting scheme for solar irradiation. In this scheme, raw training data were stored and computed locally at the clients; only the trained forecasting models were shared [17]. Venkataramanan et al. proposed a distributed machine learning approach to forecast energy from distributed resources [18]. Wang et al. [19] predict sociodemographic information of households based on the FL framework. In addition, the FL framework has been introduced to wind turbine systems for distributed modeling in the European Horizon 2020 project, including the Smart4RES [20] and PHOENIX [21]. The performance of the blade icing detection model is often related to local climatic conditions, while the climatic factors in different regions are often very different, such as humidity and light distribution. Therefore, the modeling needs to take into account the difference in climatic conditions. Nevertheless, if the model is created separately in different regions, its generalization will be poor (due to the use of only local data for training). The FL framework comes into effect, which trains local models separately but aggregates them to obtain a global model.

In the classical FL framework, each client first trains a local model based on its own data to obtain a global model. Then, each client sends its model parameters, W_l, to the server. Finally, the server obtains a global model by aggregation, e.g., by computing the weighted averaging of the model parameters, i.e., $W \leftarrow \sum_{l=1}^{L} \frac{N_l}{N} W_l$. There are two main kinds of heterogeneity in FL: data heterogeneity and model heterogeneity [22]. Data heterogeneity is also known as the non-independent and identically distributed (IID) problem. Different wind turbines may work under different weather conditions, resulting in the distribution of the collected sensor data being non-IID. Li et al. introduced the FedProx to overcome the data heterogeneity by using a local regularization term in each client model [23]. Arivazhagan et al. presented a personalized FL model, in which the local model is designed with the shape of base + personalization layer [24]. Similar works also can be found from [25]. There are also some other attempts for generating global models by clustering local models [26], [27]. To overcome the model heterogeneity, KD-based are gained more attention by distilling knowledge from the server model to client models, which have different model structures [28], [29]. Some researchers are integrating the network structure search and FL to discover customized models for clients with different computational capabilities [30], [31]. However, most of the methods above focused on a single heterogeneous challenging scenario of FL, and did not fully consider the available computing capability of each client. To address this challenge, we propose a heterogeneous collaborative FL method where the clients and the server can have different model structures. More specifically, each client will have a small model, while the server will have a large model, which can better fit the specific configuration of the clients and the server.

III. BiFL for blade icing detection of wind turbine

A. Overview

Wind farms are usually located in remote areas and do not have powerful computing capabilities, such as computing devices with GPUs and big memory. If the traditional FL model is applied to wind turbines for blade icing detection, it is necessary to upgrade the computing facilities, resulting in additional costs. Therefore, this paper proposes a heterogeneous FL model for blade icing detection for different types of wind turbines under different weather conditions. Inspired by [32], we propose a class-imbalanced heterogeneous federated learning method, BiFL. As illustrated in Fig. 2, BiFL has K local client models and a global server model. The clients at different locations (in different climate zones) can use their data to train their local models. The knowledge gained from each client model is then uploaded to the central server to obtain a global model through aggregation. The global knowledge is sent back to each client to help improve the local model. Due to the different environmental conditions in

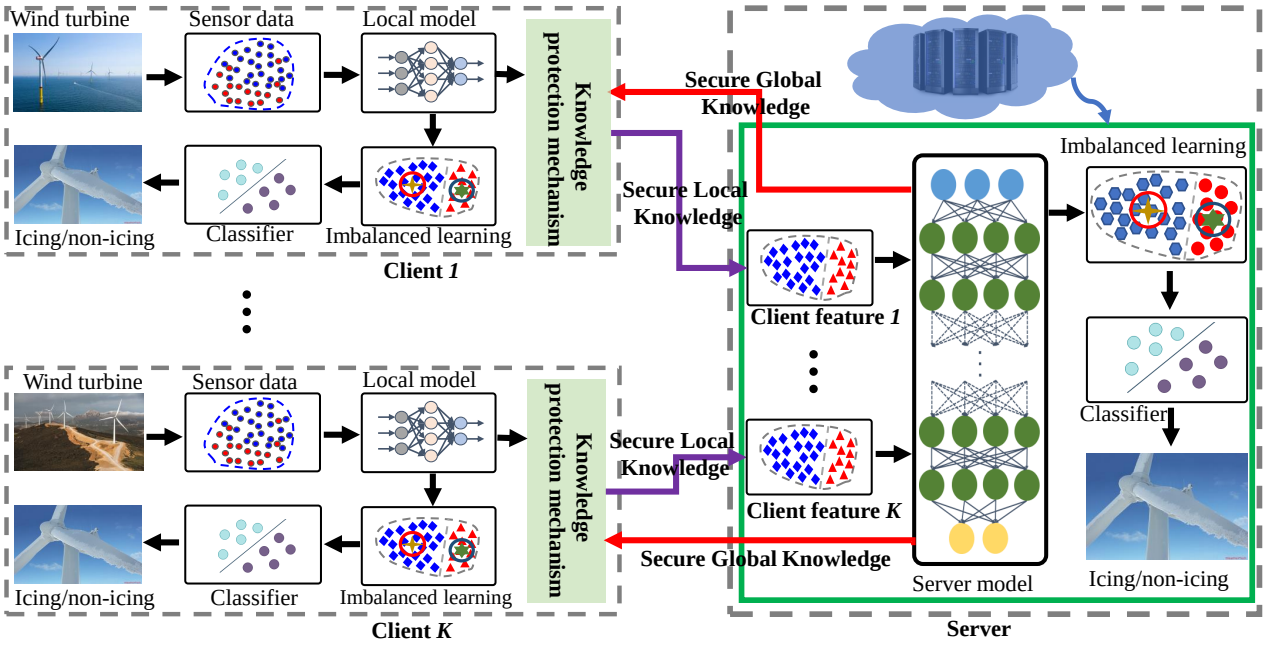

Fig. 2. Structure of the proposed BiFL

which the wind turbines are located, the sensor data collected by the SCADA system is usually characterized by a high level of non-linearity and non-stationary. In this paper, we use a convolutional neural network (CNN) on both the client and server sides to extract useful features. Due to the limited computational capacity of each client, the local models of the clients are usually much simpler than the global model of the server. Therefore, we train a small CNN on each client and a large specially designed CNN in the central server. The local client and global server models can then exchange knowledge periodically.

B. Feature extractor for client and server model

Since the clients and the server usually have different computational capabilities, their models must have different complexities. In this paper, a single-layer CNN (SLCNN) is proposed for the client model, and a specially designed densely connected CNN is proposed for the server model. Figure 3 and 4 describe the structures of the two models, respectively.

The Φ_c client model directly uses the raw data and sends the encoded features to the server model for protecting the data. The client model consists of a convolution layer (Conv1D), an attention layer (SE) [33], a batch normalized layer (BN), and an activation layer (LeakyReLU). Note that the encoded features sent to the server are the output of the LeakyReLU layer. Given the raw input $X_{raw} \in \Re^{T\times d}$, d and T are the input dimension and the sample window size, respectively. The outputs of the SLCNN are represented by:

$$\begin{aligned} X_c &= Conv1D(X_{raw}^T) \\ X_{SE} &= SE(X_c) \\ X_{SLCNN} &= LeakyReLU(BN(X_{SE})) \end{aligned} \tag{1}$$

where X_c, X_{SE} and X_{SLCNN} are the outputs of the Conv1D, SE, and LeakyRe layers, respectively. The outputs have the same shape, $\Re^{F\times L}$, where F is the number of filters in the CNN, L is the length of encoded feature map. To achieve knowledge

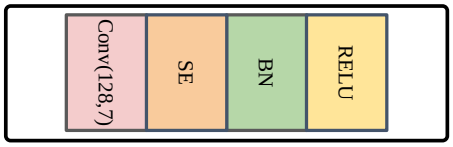

Fig. 3. Neural network architecture of client model.

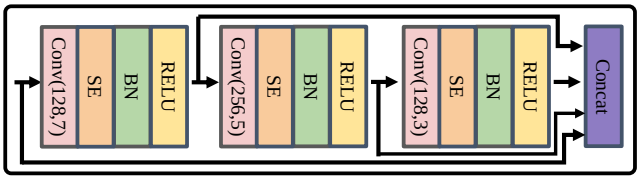

Fig. 4. Neural network architecture of server model.

transfer, BiFL sends the extracted features, X_{SLCNN}, predicted labels, Y_p, and true labels, Y_t, to the server.

The inputs to the server model, Φ_s, are the local features extracted from the client models. Since the number of client models can be many, we use a densely connected CNN as the server model to capture the information from the client models. The server model consists of three convolutional blocks, X_1, X_2, and X_3, each of which is identical to the convolutional block in the client model. This structure allows for hierarchical information learning through convolutional operations and dense connections. The output of the server model is presented as $X_s = [X_f, X_1, X_2, X_3]$, where $[\cdot,\cdot]$ represents concatenation and X_f represents the features extracted from the client models.

C. Prototype method for class imbalanced data

Class imbalance in the training data can compromise the performance of the resulting models, and in particular, affect the classification ability of minority classes. The imbalance problem can be addressed at two levels: at the data level and at the algorithm level. Data-level approaches, as the

name implies, require some preprocessing of the raw data, such as resampling and data enrichment, to increase the number of minority classes or reduce the number of majority classes [34]. Data-level approaches also require additional information in the modeling, such as the data distribution. Since this may violate data privacy requirements, they are not suitable for the FL framework [35]. In contrast, algorithm-level approaches focus on improving the training algorithm or tuning the training parameters [36], [37]. Algorithm-level approaches require no (or less) data preprocessing. Therefore, algorithm-level approaches are more suitable for FL, although many hyperparameters are needed for tuning. In this paper, we propose a prototype-based method to address the data imbalance problem in LF. Instead of sampling the raw data, the idea is to balance the features through a neural network and adjust its hyperparameters. Since the raw data is imbalanced in this study, the latent features extracted by the neural network are also imbalanced. To balance the features/classes, we first learn a prototype for each class to the latent space, which is obtained by feature extraction. We then build a classifier based on the obtained prototypes. Therefore, the proposed method does not require information about the distribution of the raw data, and thus effectively preserves privacy. In this study, we adopt the attention-oriented prototype learning method [38], which makes use of important information while ignoring irrelevant information.

The prototype is a vector presenting each class. The simplest way to compute the prototype is to average all features in the latent space:

$$C^i = \frac{1}{n^i}\sum_{k=1}^{n^i} X^{i,k}, \tag{2}$$

where $C^i \in \mathbb{R}^{1\times H}$ is the prototype for a class, $i \in [1, \cdots, n_c]$, n_c is the total number of the classes, n^i is the number of instances in the deep latent space for class i, and X represents the features of the samples in the deep latent space.

Although the calculation of the prototype based on Equation (2) is effective, it is necessary to focus on the important features while ignoring the irrelevant ones. Therefore, we employ an attentional prototype (AP) [38], defined as follows. Let $X_k = [x_1, \ldots, x_m] \in \Re^{m\times dl}$ be a matrix of latent features for k, m is the total number of data samples with class label k, and dl is the output dimension of the client and the server models.

$$e_k = sigmoid(U_k^T tanh(V_k X_k^T)), \tag{3}$$

where $U_k \in \Re^{w\times 1}$ and $V_k \in \Re^{w\times dl}$ are the trainable parameters of the prototype attention module; w is the dimension of the two attention parameters.

Then, the attention weight can be calculated as follows:

$$\alpha_k = \frac{exp(e_k)}{\sum_{j=1}^{m} exp(e_k^j)}, \tag{4}$$

When the attention weight is calculated, the attentional prototype of the class k can be obtained by:

$$C_k = \alpha_k * X_k, \tag{5}$$

Algorithm 1 The training process of the server model

Input: Training epoch E_s, communication rounds R, the number of clients K, server model Φ_s

Server:

Initialize the model parameters W_s of the server model Φ_s
for *each communication round r to R* **do**
  for *each client k to K* **do**
    X_c^k, $y_t^k \leftarrow$ Client(r, k)
    $F[k] \leftarrow X_c^k$
    $Y_t[k] \leftarrow y_t^k$
  end
  for $e = 0;\ e < E_s;\ e = e + 1$ **do**
    for *each client k to K* **do**
      $X_s^k \leftarrow \Phi_s(F[k])$ ▷ features of the k-th client
      for *each class i* **do**
        $X_i^k \leftarrow X_s^k(idx = i)$ ▷ matrix for class i
        $e_i^k \leftarrow sigmoid((U_i^k)^T tanh(V_i^k (X_i^k)^T))$
        $\alpha_i^k \leftarrow \frac{exp(e_i^k)}{\sum_{j=1}^{m} exp((e_i^k)^j)}$
        $C_i^k \leftarrow \alpha_i^k * X_i^k$ ▷ prototypes
      end
      $p_s^k \leftarrow \frac{exp(-dist(\Phi_s(X), C_i^k))}{\sum_n exp(-dist(\Phi_s(X), C_n))}$ ▷ probability
      Sending p_s^k to the k-th client
      $l\_s \leftarrow L_{CE}(Y_t[k], p_s^k)$ ▷ CE loss (Eq. (6))
      $W_s \leftarrow W_s - \eta_s \nabla l_s(W_s)$ ▷ parameter update
    end
  end
end

In this paper, we use separate parameters, U_k and V_k, for each class because different classes may have different levels of attention in their feature spaces. The imbalanced features are compensated for by Equation (5).

D. Knowledge transfer enhanced by KD

To train the classifier, we use the binary cross-entropy loss defined as:

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)), \tag{6}$$

where y_i is the true value for the i-th class; $p(\cdot)$ is the estimated probability for the i-th class; and N is the number of samples.

To estimate the probability for the i-th class, we use the squared Euclidean distance between the i-th class and a time series, defined as:

$$p(y_i) = \frac{exp(-dist(\Phi(X), C_i))}{\sum_k exp(-dist(\Phi(X), C_k))}, \tag{7}$$

where Φ is the neural network (i.e., the neural network of the client and server models) and C_i is the prototype of the i-th class.

In this paper, all clients send their learned knowledge to the server, and the server sends the aggregated information back to each client. KD enhances knowledge transfer from the server model to each client model. Since the number of clients can be very large, the knowledge transfer from clients to the server

Algorithm 2 The training process of the client model

Input: Training epoch E_c, α, client model Φ_c

Client $(r,\ k)$**:**

Initialize model parameters W_c of model Φ_c

for $e = 0;\ e < E_c;\ e = e + 1$ **do**

$X_c \leftarrow \Phi_c(X_{raw})$ ▷ features from raw sensor data

for *each class* i **do**

$X_i \leftarrow X_c(idx = i)$ ▷ matrix for class i

$e_i \leftarrow sigmoid(U_i^T tanh(V_i X_i^T))$

$\alpha_i \leftarrow \frac{exp(e_i)}{\sum_{j=1}^{m} exp(e_i^j)}$

$C_i \leftarrow \alpha_i * X_i$ ▷ prototype for each class

end

$p_c \leftarrow \frac{exp(-dist(\Phi_c(X), C_i))}{\sum_k exp(-dist(\Phi_c(X), C_k))}$ ▷ probability

$l_{ce} \leftarrow L_{CE}(y_t, p_c)$ ▷ classification loss (Eq. (6))

$l\_kd \leftarrow L_{KD}(p_c, p_s^k)$ ▷ KD loss (Eq.(8))

$l_c \leftarrow \alpha * l\_ce + (1-\alpha) l\_kd$ ▷ total loss (Eq. (9))

$W_c \leftarrow W_c - \eta_c \nabla l_c(W_c)$ ▷ parameter update

end

return X_c, y_t

does not use KD, unlike to other methods, such as [39] that require a public dataset. KD is implemented directly using the predicted value of each client model and the predicted value of the server model.

$$L_{KD} = KL(p_c||p_s) = \frac{1}{K}\sum_{i}^{N} p_c log(\frac{p_c}{p_s}), \tag{8}$$

where p_c and p_s are the predicted values of a client model and the server model, respectively.

BiFL uses the following loss function that combines the above two losses for the training:

$$L_{all} = \alpha L_{CE} + (1-\alpha) L_{KD}, \tag{9}$$

where α is the hyperparameter for balancing the cross-entropy loss and the KD loss, and set to 0.9 in our experiments.

Algorithm 1 and 2 describe the training process of the client and server models, respectively.

E. Knowledge protection mechanism

As mentioned earlier, it is necessary to exchange the learned feature maps and labels between clients and the central server over the network, which is vulnerable to cyber-attacks and leads to data leakage. To address this, we introduce the knowledge protection mechanism, as shown in Figure 2. This knowledge protection mechanism can generate obfuscated feature maps and labels that are less correlated with the original feature maps to prevent adversary reconstructions [40].

Given K client models (i.e., data owners) $l_1, l_2, \cdots, l_K$ and a global server model S, each client uses its own data $X_{raw}^i (i \in \{1, \cdots, K\})$ to train a local deep neural network $\Omega_i (i \in \{1, \cdots, K\})$. $X_{raw}^i \in \Re^{n \times T_i \times d_i}$ has n training samples, and the shape of each sample is determined by each client. To speed up the training, the n training samples are reorganized into B_i batches. Note that the T_i, and d_i $(i \in \{1, \cdots, K\})$ are confidential to other participants. Assuming that the raw learned feature map of the i-th client is $X_h^i \in \Re^{B_i \times F \times L}$, where F and L are the same to all clients, the knowledge protection mechanism can be mathematically formulated as follows:

$$X_{Enc}^i = \underbrace{Enc(\cdots Enc(Enc}_{e-1}(X_h^i, X_{h1}^i, \beta_1), X_{h2}^i, \beta_2), X_{h3}^i, \beta_3), \cdots, X_{he}^i, \beta_e) \tag{10}$$

$$Y_{Enc}^i = \underbrace{Enc(\cdots Enc(Enc}_{e-1}(Y^i, Y_1^i, \beta_1), Y_2^i, \beta_2), Y_3^i, \beta_3), \cdots, Y_e^i, \beta_e) \tag{11}$$

where X_{hj}^i and Y_j^i $(j \in \{1, \cdots, e\})$ are the encrypted feature map and the corresponding label. $\beta_j \sim Beta(1,1)$ is the encoding factor. The *Enc* operation is defined by:

$$Enc(X_1, X_2, \beta) = \beta \cdot X_1 + (1-\beta) \cdot X_2 \tag{12}$$

With this data protection mechanism, each client sends the encoded feature map securely to the server. Note that the feature maps sent to the server must have the same shape, which may be the output of an arbitrary hidden layer. Due to the heterogeneity of client models (their structures may be different) in BiFL, we can assume that all participants have no information about each other's model structures, further protecting the data from feature-based inference attacks. In this paper, the knowledge exchange between clients and the server occurs only during the training phase. Therefore, the risk of data reconstruction becomes difficult because the attacker's access can only be limited to the evolving and untrained feature maps, rather than the fully trained feature maps that represent the raw data [32]. The feature maps will be eliminated once the server has generated the global model.

BiFL's design can help prevent peer inference attacks between clients. This is because BiFL adopts a non-sharing architecture, in which each client keeps its data secure. However, if one client can perform inference attacks and pattern memory attacks on other clients, then protection methods, such as secure multi-party computation, homomorphic encryption, blockchain, or differential privacy, can be applied to improve data security.

F. Computational complexity analysis

The computational intensive components of the client and the server models include the Conv1D layer, SE layer and prototype learning module, and their computational complexity can be defined as $\mathcal{O}(F \cdot k \cdot d \cdot T)$, $\mathcal{O}(F^2)$ and $\mathcal{O}(w \cdot dl \cdot m)$, where F is the number of filters, k is the size of the filter, d is the number of features, and T is the length of the time series data. The w, m, and dl are three parameters for the attentional prototype, where w is the size of these two trainable parameters, m is the number of feature samples, and dl is the feature dimension. The computational complexity of the knowledge protection mechanism is $\mathcal{O}(e)$, and it can be ignored due to e is small (< 4). The combined complexity of the server and client models can then be defined as: $\mathcal{O}(n_s \cdot (F_s \cdot k_s \cdot d_s \cdot T_s + F_s^2) + w_s \cdot dl_s \cdot m_s)$ and $\mathcal{O}(F_c \cdot k_c \cdot d_c \cdot T_c + F_c^2 + w_c \cdot dl_c \cdot m_c)$, where n_s is the

number of layers in the server model and the subscripts s and c represent the client and the server, respectively. Given the data size D for each client, the overall complexity of the BiFL becomes: $\mathcal{O}[R \cdot (E_s \cdot K \cdot D \cdot ((n_s \cdot (F_s \cdot k_s \cdot d_s \cdot T_s + F_s^2) + w_s \cdot dl_s \cdot m_s) + E_c \cdot K \cdot D \cdot (F_c \cdot k_c \cdot d_c \cdot T_c + F_c^2 + w_c \cdot dl_c \cdot m_c)))]$, where R is the communication round between the server and all clients, K is the number of the clients, and E_s and E_c are the training epochs of the server and the clients, respectively.

IV. Experiment

We use Pytorch (v.1.8.0) to implement the models. All experiments were performed on a server with a 16GB Tesla T4. The following hyper-parameters are used for training: the learning rate of the Adam optimization is $1e-3$; the learning epochs of the client and the server are $E_c = 5$ and $E_s = 10$, respectively; the number of the communication rounds between clients and the server, R, is 20.

A. Experimental settings and data

The experimental data are from two wind farms in China's Shanxi and Henan Provinces, located in northern China, about 700 kilometers apart. We simulated 20 wind turbines as clients, 10 per wind farm, and a central server for aggregation. The data were collected from 11 February 2019 to 26 February 2019 for one wind farm and from 12 February 2019 to 26 February 2019 for the other. The resolution of the data is 30 seconds.

Wind turbine experts identified 16 variables related to blade icing, as shown in Table I, and labeled the data as icing or normal. The raw data quality includes noise, outliers, and missing values, cleaned up prior to modeling. The data were normalized to scalar values to minimize the impact of unity, and the raw time series data were segmented into a fixed-length (or window size).

TABLE I
SCADA Data Specification

No.	Variable name	Description
1	wind_speed	Wind speed
2	wind_direction	Wind direction
3	generator_speed	Generator speed
4	power	Active power
5	yaw_position	Yaw position
6	pitch1_angle	Angle of pitch 1
7	pitch2_angle	Angle of pitch 2
8	pitch3_angle	Angle of pitch 3
9	pitch1_speed	Speed of pitch 1
10	pitch2_speed	Speed of pitch 2
11	pitch3_speed	Speed of pitch 3
12	environment_temp	Environment temperature
13	internal_temp	Internal temperature of nacelle
14	pitch1_moto_tmp	Temperature of pitch motor 1
15	pitch2_moto_tmp	Temperature of pitch motor 2
16	pitch3_moto_tmp	Temperature of pitch motor 3

60% of the data were used for training for each client, and the remaining 40% for testing. To assess the robustness of the global model, we tested three different imbalance ratios, i.e., the ratio between the number of normal classes and the number of icing classes. The imbalance ratio ρ is defined as $\rho = \frac{N_{normal}}{N_{Icing}}$, where N_{normal} and N_{Icing} are the number of samples labeled with the normal and icing class, respectively. The imbalance ratio of the test data is 10:1.

B. Evaluation metrics

The following two metrics are used, F_β for classification and BA (balanced-accuracy) for imbalanced learning, defined as follows:

$$F_\beta = \frac{(1+\beta^2) \times Precision \times Recall}{\beta^2 Precision + Recall} \tag{13}$$

$$Precision = \frac{TP}{TP+FP} \quad Recall = \frac{TP}{TP+FN} \tag{14}$$

$$\texttt{BA} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{15}$$

where TP, FP, FN, and TN represent true positive, false positive, false negative, and true negative, respectively. The value of β in F_β was set to 2 in the experiment.

For K client models, the average values of the metrics are:

$$mF_\beta = \frac{1}{K}\sum_i^K F_\beta^i \quad mBA = \frac{1}{K}\sum_i^K BA^i \tag{16}$$

To minimize the effect of randomness, we averaged the values of all server training rounds in each communication.

C. Comparison with state-of-the-art FL methods

TABLE II
Performance comparison with federated learning methods

Method	$\rho = 20:1$		$\rho = 50:1$		$\rho = 100:1$	
	mF_β	mBA	mF_β	mBA	mF_β	mBA
FedAvg	45.56	70.01	38.64	64.37	37.95	63.74
FLGKT	**80.14**	**91.05**	61.28	80.70	58.02	76.43
Ours	77.72	90.25	65.76	84.77	64.76	83.82

We have introduced the following two FL frameworks for our evaluation. 1) **FedAvg** (federated mean) [7], which is a classic and standard FL framework. FedAvg consists of multiple clients and a server. Clients use local data to train their models and upload them to the server; the server initializes the network at the start of training and aggregates the client models using a weighted average method. FedAvg requires that the server and clients share the same network architecture. 2) **FLGKT** (FL as group knowledge transfer) [32], which is a state-of-the-art FL framework for knowledge transfer from clients to the server. The clients use a lighter architecture model than the server to reduce the computing workload.

The sampling window size is 128 (approximately 11 minutes). In BiFL, the number of CNN filters is set to 128. The α is set to 0.9 and the *m* is 2. TABLE II shows the results where the best value for each metric is in bold. The results show that BiFL achieves the best performance, except for $\rho = 20:1$, which ranks second, lower than FLGKT. However, FLGKT's client and server models were more complex than BiFL's. This is the main reason why it outperforms ours when ρ =20:1. When $\rho = 50:1$, the mF_β and mBA of our model improves by 41.24% and 24.07%, respectively, over FedAvg, and are comparable to FLGKT. When $\rho = 100:1$, the mF_β and mBA of our model reach 41.40% and 23.96% better than FedAvg, respectively. Compared to FLGKT, our model shows

an improvement of 10.41% and 8.82% for mF_β and mBA, respectively. We can also observe that as the imbalance rate increases, the accuracy of all three methods decreases.

D. Comparison with state-of-the-art class-imbalance methods

TABLE III
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART CLASS-IMBALANCED METHODS

Method	$\rho = 20:1$		$\rho = 50:1$		$\rho = 100:1$	
	mF_β	mBA	mF_β	mBA	mF_β	mBA
Focal	25.74	55.38	21.68	53.18	24.82	54.41
WCE	45.11	69.49	44.49	69.04	43.70	68.12
CB	44.55	68.88	43.91	68.37	43.09	67.42
GHMC	27.94	51.21	26.92	50.12	27.11	50.27
LDAM	18.96	52.35	16.73	51.38	16.31	50.86
Balance	72.15	88.83	65.66	**85.04**	62.22	82.36
Ours	**77.72**	**90.25**	**65.76**	84.77	**64.76**	**83.82**

We compare the proposed BiFL with the following six class-imbalanced algorithms: 1) **Focal** (focal loss): It is an often-used loss function for class-imbalanced learning. We set γ = 1 and use the same weights as in [36] for each class. 2) **WCE** (Weighted CE): This is an enhanced version of CE for imbalanced learning by considering that different classes have different weights. 3) **CB** (class-balance): It introduces the concept of the effective number of samples, taking into account data overlap. Based on the effective number of samples per class, the class balance loss is calculated [37]. 4) **GHMC** (gradient harmonizing mechanism classification): It is a newly proposed method for the data imbalanced problem that conquers the disharmony in imbalanced classification [41]. 5) **LDAM** (label-distribution-aware margin): It can replace the standard cross-entropy objective during training and be used with prior class imbalance training strategies such as re-weighting and resampling [42]. 6) **Balance**: It performs undersampling on the majority class using a data-level algorithm [34].

As shown in TABLE III, our model has the best performance in terms of mF_β and mBA, compared to Focal, WCE, CB, GHMC, and LDAM. We can also observe that WCE ranks second, and LDAM is the least in terms of mF_β in the three cases. Our model improves mF_β and mBA by 41.96% and 23.00% over WCE when $\rho = 20:1$. When $\rho = 50:1$ and $\rho = 100:1$, its performance increases by 18.56% and 18.73% for mBA, and by 32.34% and 32.52% for mF_β compared to WCE. Compared to the class balance method, our model is competitive with mF_β, but inferior to mBA in the case of $\rho = 50:1$. From the definition of mBA and mF_β, it is clear that mBA focuses on the classification accuracy of all correct samples, while mF_β computes the classification accuracy of icing samples. In other words, our method performs better than Balance for blade icing detection. In addition, balancing the data requires the collaboration of all clients, which is time and resource-consuming.

E. Impact of knowledge protection mechanism

This section will study the performance impact of the used knowledge protection mechanism. We vary the *m* from 2 to 4 and compare it with the variant, **No_KPM**, in which the knowledge protection mechanism is removed from BiFL.

From TABLE IV, We can see that adding knowledge protection mechanism causes a performance degradation for both $\rho = 20:1$ and $\rho = 50:1$. When $\rho = 100:1$, the result shows that adding the knowledge protection mechanism can slightly improve the performance for *m=3* and *m=4*. However, when *m* is increased, the changes in the model performance are irregular, i.e., it is difficult to identify the change patterns. Moreover, there is no downward trend as expected with the increase of imbalanced ratio ρ, when the knowledge protection mechanism is added. The reason is that when the protection mechanism has been added, the raw features are confused by encoding, and the confusing information might impact the prototypes. For example, it might lead to the deviation of prototypes, which results in the change in final results.

TABLE IV
PERFORMANCE IMPACTS OF KNOWLEDGE PROTECTION MECHANISM

Method	$\rho = 20:1$		$\rho = 50:1$		$\rho = 100:1$	
	mF_β	mBA	mF_β	mBA	mF_β	mBA
m = 2	77.72	90.25	65.76	84.77	64.76	83.82
m = 3	68.07	86.13	72.19	88.27	**74.43**	**88.44**
m = 4	65.21	84.85	71.78	**88.42**	72.01	87.86
No_KPM	**86.28**	**92.61**	**75.89**	86.35	68.71	82.24

F. Ablation and sensitivity analysis

We now conduct an ablation study to evaluate the AP, SE, and KD modules. The following four variants are derived from BiFL: 1) **No_AP**, in which the AP module is removed; 2) **No_AttP**, in which the attention module was removed from the AP module (i.e., the prototype calculated based on Equation (2)); 3) **No_SE**, in which the SE module was removed.

TABLE V shows the results. We can observe that the performance decreases the most when the AP module is not used in all three cases. Specifically, the performance of mF_{beta} decreases by 42.93%, 33.47% and 29.54% (absolute value); the performance of mBA decreases by 29.90%, 27.43% and 23.68%. In all three cases, when the SE module is not used, the performances decrease by 17.69%, 6.69% and 6.06% for mF_β, and by 8.96%, 3.98% and 3.58% for mBA. Comparing No_AP and No_AttP, an improvement is observed for all three cases for both metrics, validating the effectiveness of the proposed attention module in improving prototype learning. Therefore, we can safely conclude that the proposed modules and their combinations are effective based on the above studies.

TABLE V
ABLATION STUDY

Method	$\rho = 20:1$		$\rho = 50:1$		$\rho = 100:1$	
	mF_β	mBA	mF_β	mBA	mF_β	mBA
No_AP	34.79	60.35	32.29	57.34	35.22	60.14
No_AttP	52.80	75.94	51.17	74.68	44.63	69.52
No_SE	60.03	81.29	59.07	80.79	58.70	79.97
Ours	77.72	90.25	65.76	84.77	64.76	83.82

To evaluate the impact of the input window size and α, we perform the following sensitivity analysis. We vary the window size from 32 to 256, and increase α from 0.15 to 0.55, and their results are shown in Fig. 5 and 6, respectively. From Fig. 5, we can see that the performance is almost linearly proportional to the window size for both metrics (mF_β and mBA), and

it reaches its maximum value when the window size is 256. From Fig. 6, we can see that the performance of both metrics slightly decreases with increasing α when $\rho = 20:1$ and $\rho = 50:1$. When $\rho = 100:1$, the performances fluctuate slightly for both metrics, which means that the impact of α is subtle. According to Equation (9), we can know that the smaller α is, the smaller the influence of cross-entropy loss and the larger the effect of KD. This result further verifies the positive influence of knowledge transfer from the server model to the client models.

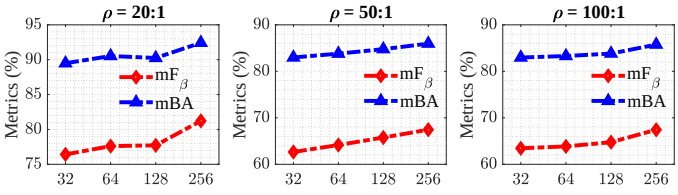

Fig. 5. Sensitivity analysis of window size

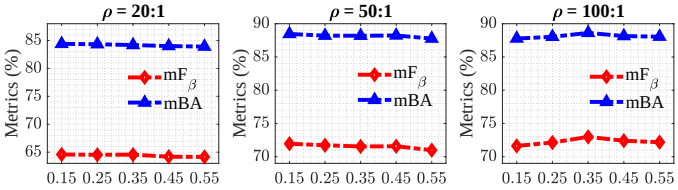

Fig. 6. Sensitivity analysis of α

V. Conclusions and Future Work

This study proposes a class imbalanced heterogeneous federated learning model, *BiFL*, for wind turbine blade icing detection. Since blade icing is strongly related to the local environment and wind turbine specifications, BiFL introduces the concept of transfer learning to improve the robustness and generalization of the model. This paper also proposed a prototype-based method to address the problem of class imbalance in the training data. Finally, this paper comprehensively evaluated the proposed model by comparing it with two classical FL models and five state-of-the-art class imbalance learning methods. The experimental results demonstrated the superiority of the proposed model. The ablation study also validated the effectiveness of the BiFL modules, and the sensitivity analysis accessed the impact of key hyperparameters.

There are several directions for future work. First, we would like to reduce the risk of data breach further, as clients need to upload the extracted features to the server to train the global model. Differential privacy and blockchain can be applied to the uploaded data and protect it from network attacks. Second, we would like to identify the severity of icing on wind turbine blades. Potential solutions may include interpreting the output probability or labeling the icing severity based on domain knowledge. Third, the depthwise convolution and other operations can be used to design a lightweight convolutional neural network with reduced calculations and a deeper network to enhance the feature learning.

References

[1] K. Wei, Y. Yang, H. Zuo, and D. Zhong, "A review on ice detection technology and ice elimination technology for wind turbine," *Wind Energy*, vol. 23, 2020.

[2] T. Laakso, I. Baring-Gould, M. Durstewitz, R. Horbaty, A. Lacroix, E. Peltola, G. Ronsten, L. Tallhaug, and T. Wallenius, "State-of-the-art of wind energy in cold climates," *English*, vol. 152, 09 2010.

[3] C. Yin, Z. Zhang, Z. Wang, and H. Guo, "Numerical simulation and experimental validation of ultrasonic de-icing system for wind turbine blade," *Applied Acoustics*, vol. 114, pp. 19–26, 2016.

[4] L. Shu, G. Qiu, Q. Hu, X. Jiang, G. McClure, and H. Yang, "Numerical and field experimental investigation of wind turbine dynamic de-icing process," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 175, pp. 90–99, 2018.

[5] X. Cheng, F. Shi, M. Zhao, G. Li, H. Zhang, and S. Chen, "Temporal attention convolutional neural network for estimation of icing probability on wind turbine blades," *IEEE Transactions on Industrial Electronics*, pp. 1–1, 2021.

[6] C. Goncalves, R. J. Bessa, and P. Pinson, "Privacy-preserving distributed learning for renewable energy forecasting," *IEEE Transactions on Sustainable Energy*, 2021.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[8] M. Zhang, J. Yu, R. Chen, Q. Liu, J. Liu, D. Song, P. Liu, L. Gao, and J. Wang, "Highly transparent and robust slippery lubricant-infused porous surfaces with anti-icing and anti-fouling performances," *Journal of Alloys and Compounds*, vol. 803, pp. 51–60, 2019.

[9] C. Peng, S. Xing, Z. Yuan, J. Xiao, C. Wang, and J. Zeng, "Preparation and anti-icing of superhydrophobic pvdf coating on a wind turbine blade," *Applied Surface Science*, vol. 259, pp. 764–768, 2012.

[10] J. Zeng and B. Song, "Research on experiment and numerical simulation of ultrasonic de-icing for wind turbine blades," *Renewable Energy*, vol. 113, pp. 706–712, 2017.

[11] M. L. Corradini, A. Cristofaro, and S. Pettinari, "A model-based robust icing detection and estimation scheme for wind turbines," in *2016 European Control Conference (ECC)*. IEEE, 2016, pp. 1451–1456.

[12] N. Jolin, D. Bolduc, N. Swytink-Binnema, G. Rosso, and C. Godreau, "Wind turbine blade ice accretion: A correlation with nacelle ice accretion," *Cold Regions Science and Technology*, vol. 157, pp. 235–241, 2019.

[13] A. A. Jiménez, F. P. G. Márquez, V. B. Moraleda, and C. Q. G. Muñoz, "Linear and nonlinear features and machine learning for wind turbine blade ice detection and diagnosis," *Renewable Energy*, vol. 132, pp. 1034–1048, 2019.

[14] Y. Liu, H. Cheng, X. Kong, Q. Wang, and H. Cui, "Intelligent wind turbine blade icing detection using supervisory control and data acquisition data and ensemble deep learning," *Energy Science & Engineering*, vol. 7, no. 6, pp. 2633–2645, 2019.

[15] B. Yuan, C. Wang, F. Jiang, M. Long, P. S. Yu, and Y. Liu, "Waveletfcnn: A deep time series classification model for wind turbine blade icing detection," *arXiv preprint arXiv:1902.05625*, 2019.

[16] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Addressing class imbalance in federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 10 165–10 173.

[17] X. Zhang, F. Fang, and J. Wang, "Probabilistic solar irradiation forecasting based on variational bayesian inference with secure federated learning," *IEEE Transactions on Industrial Informatics*, 2020.

[18] V. Venkataramanan, S. Kaza, and A. M. Annaswamy, "Der forecast using privacy preserving federated learning," *arXiv preprint arXiv:2107.03248*, 2021.

[19] Y. Wang, I. L. Bennani, X. Liu, M. Sun, and Y. Zhou, "Electricity consumer characteristics identification: A federated learning approach," *IEEE Transactions on Smart Grid*, 2021.

[20] G. Kariniotakis, S. Camal, D. van der Meer, P. Pinson, G. Giebel, L. Han, R. Bessa, Q. Libois, M. Cassas, B. Alonzo *et al.*, "Research directions and results in the smart4res project for improving renewable energy forecasting," in *Wind Energy Science Conference (WESC)*, 2021.

[21] Phoenix Project, 2020, https://phoenix-h2020.eu/.

[22] Y. Tan, G. Long, L. Liu, T. Zhou, Q. Lu, J. Jiang, and C. Zhang, "Fedproto: Federated prototype learning across heterogeneous clients," in *AAAI Conference on Artificial Intelligence*, 2022.

[23] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *MLSys*, 2020.

[24] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *arXiv preprint arXiv:1912.00818*, 2019.

[25] P. P. Liang, T. Liu, L. Ziyin, R. Salakhutdinov, and L.-P. Morency, "Think locally, act globally: Federated learning with local and global

representations," *Advances in Neural Information Processing Systems*, 2020.

[26] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," in *Advances in Neural Information Processing Systems*, 2020.

[27] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE transactions on neural networks and learning systems*, 2020.

[28] D. Li and J. Wang, "Fedmd: Heterogenous federated learning via model distillation," in *Advances in Neural Information Processing Systems*, 2020.

[29] G. Long, T. Shen, Y. Tan, L. Gerrard, A. Clarke, and J. Jiang, "Federated learning for privacy-preserving open innovation future on digital health," *arXiv preprint arXiv:2108.10761*, 2021.

[30] C. He, M. Annavaram, and S. Avestimehr, "Fednas: Federated deep learning via neural architecture search," in *CVPR 2020 Workshop on Neural Architecture Search and Beyond for Representation Learning*, 2020.

[31] I. Singh, H. Zhou, K. Yang, M. Ding, B. Lin, and P. Xie, "Differentially-private federated neural architecture search," in *FL-International Conference on Machine Learning Workshop*, 2020.

[32] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large cnns at the edge," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 14 068–14 080.

[33] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[34] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 1, pp. 1–30, 2018.

[35] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Towards class imbalance in federated learning," *arXiv preprint arXiv:2008.06217*, 2020.

[36] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[37] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9268–9277.

[38] X. Zhang, Y. Gao, J. Lin, and C.-T. Lu, "Tapnet: Multivariate time series classification with attentional prototypical network." in *AAAI*, 2020, pp. 6845–6852.

[39] W. Zhuang, Y. Wen, X. Zhang, X. Gan, D. Yin, D. Zhou, S. Zhang, and S. Yi, "Performance optimization of federated person re-identification via benchmark analysis," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 955–963.

[40] D. Xiao, C. Yang, and W. Wu, "Mixing activations and labels in distributed training for split learning," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[41] B. Li, Y. Liu, and X. Wang, "Gradient harmonized single-stage detector," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 8577–8584.

[42] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," in *Advances in Neural Information Processing Systems*, 2019, pp. 1567–1578.