Secure Transfer Learning for Machine Fault Diagnosis under Different Operating Conditions*

Chao Jin[1][0000−0002−6858−1177], Mohamed Ragab[2][0000−0002−2138−4395], and Khin Mi Mi Aung[1][0000−0002−5652−3455]

[1] Institute for Infocomm Research, A*STAR, Singapore
{jin_chao, mi_mi_aung}@i2r.a-star.edu.sg
[2] School of Computer Science and Engineering, Nanyang Technological University
mohamedr002@e.ntu.edu.sg

Abstract. The success of deep learning is largely due to the availability of big training data nowadays. However, data privacy could be a big concern, especially when the training or inference is done on untrusted third-party servers. Fully Homomorphic Encryption (FHE) is a powerful cryptography technique that enables computation on encrypted data in the absence of decryption key, thus could protect data privacy in an outsourced computation environment. However, due to its large performance and resource overheads, current applications of FHE to deep learning are still limited to very simple tasks. In this paper, we first propose a neural network training framework on FHE encrypted data, namely PrivGD. PrivGD leverages the Single-Instruction Multiple-Data (SIMD) packing feature of FHE to efficiently implement the Gradient Descent algorithm in the encrypted domain. In particular, PrivGD is the first to support training a multi-class classification network with double-precision float-point weights through approximated Softmax function in FHE, which has never been done before to the best of our knowledge. Then, we show how to apply FHE with transfer learning for more complicated real-world applications. We consider outsourced diagnosis services, as with the Machine-Learning-as-a-Service paradigm, for multi-class machine faults on machine sensor datasets under different operating conditions. As directly applying the source model trained on the source dataset (collected from source operating condition) to the target dataset (collect from the target operating condition) will lead to degraded diagnosis accuracy, we propose to transfer the source model to the target domain by retraining (fine-tuning) the classifier of the source model with data from the target domain. The target domain data is encrypted with FHE so that its privacy is preserved during the transfer learning process. We implement the secure transfer learning process with our PrivGD framework. Experiments results show that by fine-tuning a source model for fewer than 10 epochs with encrypted target domain data, the model can converge to an increased diagnosis accuracy by up to 20%, while the whole fine-tuning process takes approximate 3.85 hours on our commodity server.

* This research / project is supported by A*STAR under its RIE2020 Advanced Manufacturing and Engineering (AME) Programmatic Programme (Award A19E3b0099)

Keywords: Homomorphic Encryption · Data Privacy · Transfer Learning · Fault Diagnosis.

1 Introduction

Machine Learning as a Service (MLaaS) is becoming an increasingly hot topic in recent years. In this paradigm, large organisations with large amounts of data can train high quality models, and share their models with other users who do not own enough data or cannot afford to train complete models of their own. This is extremely useful when the data of interest is hard to acquire. For instance, in the healthcare domain, to train a deep learning model that can predict a rare disease from a patient's X-Ray image, we need enough amount of positive samples, i.e., patents' X-Ray images with that particular disease. While a large hospital may possess enough data to train a good model, individual clinics may not have the data and resources to do so. It is therefore beneficial for the hospital to put its model on the cloud and provide inference services for the clinics. Another example is the machine fault diagnosis in the advanced manufacturing domain. It usually requires time and efforts to collect enough sensor data under machine faulty conditions to train an accurate fault diagnosis model, especially when there are multiple different failure types and different operating environments. Take the training of a simple logistic regression model as an example, it is suggested that at least $N = 10 \cdot k/p$ training samples are required, where k is the number of covariates (independent variables), and p is the smallest of the proportions of negative or positive cases in the dataset [1].

Although MLaaS enables model owners to share the usage of their models without transferring them to the users, it poses data privacy risks on the users' side, as the users need to upload their private data to the outsourced inference servers. In order to solve the data privacy issue, researchers have proposed numbers of privacy-preserving neural network inference solutions, based on cryptography technologies like Fully Homomorphic Encryption (FHE) [2–9], Multi-Party Computation (MPC) [10, 11], or hybrid of FHE and MPC [12, 13]. In particular, FHE [14] provides strong crypto primitives that enable computation directly on encrypted data. To apply FHE in the MLaaS scenario, a model is pre-trained on the clear data and deployed on an inference server, a user encrypts his data using a FHE scheme before sending the data to the inference server, and then the encrypted user data is evaluated homomorphically with the model on the inference server, and finally the inference result which is also in encrypted form is sent back to the user for decryption. FHE based solutions are considered as non-interactive, in which the server can independently evaluate the whole model and generate the predicted result. On the other hand, MPC based solutions, built on top of techniques like Garbled Circuits [15] and Secrete Sharing [16], require interactive communications between the user and the server, and considerable amount of computational load at the user side, which may not be the optimal case in many application scenarios.

Despite of its strong crypto primitives, FHE in its current state cannot be directly applied to large and deep neural network models, due to its large computational and memory resource overhead, as well as the noise growth along with computational depth. Therefore, current FHE-based solutions are only targeting for simple inference tasks like MNIST [2–5] and CIFAR10 [3, 4, 8], and even simpler training tasks like logistic regression [17].

In this paper, we apply FHE to the MLaaS scenario with a real-world application, machine fault diagnosis on the vibration sensor data. Furthermore, the sensor data may be collected under different machine operating conditions. We first assume that a model owner who possesses enough data trains a complete fault diagnosis model (including a feature extractor and a classifier) for a certain operating condition. Then the model owner deploys his model on a server and provides inference services for other users. On the other hand, the users who could not afford to train their own models can encrypt their own sensor data using FHE and send to the server for inference. However, multiple challenges may be faced here. First, the model may be too big and cannot be evaluated efficiently as a whole in the FHE domain. Second, the user data may be collected in a different machine operating conditions which may results in lower inference accuracy if directly apply the model on it. To address these challenges, we propose to use a transfer learning approach: 1) the model owner shares the feature extractor of his model among the users for them to extract the common features from their time-series sensor data; and 2) the user extracts and encrypts the features from his own data and send to the inference server to fine-tune the classifier; and finally 3) the fine-tuned classifier can be used for fault diagnosis for the user's new incoming data. Noted that we leverage transfer learning in two ways here. First, it enables sharing the common part (i.e., feature extractor) and protecting only the task-specific part (i.e., classifier) of the model, thus can significantly reduce the network size in the FHE domain. Second, the fine-tuning process with user's private data leverages on the prior knowledge (weights and biases) of the source model's classifier, thus can converge to an increased accuracy with less data and fewer number of training iterations (compared to training from scratch).

The fine-tuning process of the fault diagnosis model involves training of a multi-class classifier in the FHE domain. While training a binary classifier (logistic regression) with FHE may be an easier task, training a multi-class classifier is a much harder problem as it requires implementation of Softmax activation function in the FHE domain. To enable efficient neural network training on FHE encrypted data, we design and implement the PrivGD framework. PrivGD supports the multi-class classifier training through approximating an estimated Softmax function in the FHE domain. Moreover, PrivGD offers a more parallelized Mini-Batch Gradient Descent training procedure, by designing more efficient matrix multiplications on the encrypted data based on the powerful SIMD packing features [18] of modern FHE schemes.

The major contributions of our paper are summarized as follows.

– We design and implement PrivGD, a secure neural network training framework on FHE encrypted data. PrivGD offers optimized Mini-Batch Gradient

Descent training with FHE, and is the first to support secure training of double-precision float-point networks for multi-class classification tasks, to the best of our knowledge.
- We propose a new paradigm of privacy-preserving MLaaS based on transfer learning, where a user can use his private data to fine-tune the classifier model for personalized inference services with improved accuracy.
- We demonstrate the efficiency of our secure transfer learning paradigm on a real-world application, machine fault diagnosis through sensor data under different operating conditions. By using PrivGD, one diagnosis model for a source condition can be fine-tuned with encrypted data from a target condition, to achieve improved accuracy by up to 20% on the target condition through fewer than 10 training epochs.

The rest of the paper is organized as follows. The next section introduces preliminaries and background knowledge. In Section 3 we describe PrivGD, our neural network training framework on FHE encrypted data. In Section 4 we describe our secure transfer learning paradigm with the real-world application of machine fault diagnosis, and the experiment results are discussed in Section 5. In Section 6 we discuss about related work and finally we conclude the paper in Section 7.

2 Preliminaries

2.1 Fully Homomorphic Encryption

Since its first introduction by Rivest et al [19], FHE has always been an intriguing technology due to its ability of computing on encrypted data in the absence of the decryption key. In 2009, Gentry proposed the first construction of a FHE scheme [14]. Since then, this field has been seen great advancements and a number of new FHE schemes have been proposed [20–22]. Generally, the FHE plaintext and ciphertext spaces are polynomial rings. FHE is instantiated to preserve the algebraic structure between plaintext and ciphertext, and provides the user with two main computational operations: homomorphic addition and homomorphic multiplication. These operations can manipulate ciphertexts and produce encrypted results that are equivalent to the corresponding plaintext results after decryption.

Modern FHE schemes conceal plaintext messages with noise that can be identified and removed with the secret key [23]. As we compute on encrypted data, the noise magnitude accumulated in a ciphertext increases at a certain rate (high rate for multiplication and low rate for addition). As long as the noise is below a certain threshold, that depends on the encryption parameters, decryption can filter out the noise and retrieve the plaintext message successfully. Although FHE schemes include a primitive (known as *bootstrapping*) to refresh the noise [14] inside ciphertexts, it is extremely computationally expensive. Instead, one can use a *levelled* FHE scheme [20] that allows evaluating circuits of multiplicative depth below a certain threshold, which can be controlled by the encryption parameters.

In this way, one can avoid the expensive bootstrapping operations by selecting the appropriate encryption parameters that can accommodate the computational needs of the applications.

Next we briefly introduce a levelled FHE scheme we use in this paper, the CKKS scheme [22]. The plaintext and ciphertext are ring elements of a polynomial ring $R_q = Z_q[X]/(X^N + 1)$, where $X^N + 1$ is the polynomial modulus with degree N and $Z_q[x]$ is the polynomial with integer coefficients based on modulus q. In particular, q is the product of a group of prime factors, where the number of primes is called the "*level*" of the ciphertext. When the input data is first encrypted, its ciphertext is at the highest level, say level L. Then along with the computations, the ciphertext may gradually move down to lower levels, by removing one prime factor from q at a time. In General, L determines the largest multiplication depth a single ciphertext can have.

CKKS supports standard FHE primitives like *encode* and *decode*, *encrypt* and *decrypt*, *addition* and *multiplication* (with both ciphertexts and plaintexts). Besides that, a unique feature of CKKS is that it supports fixed-point arithmetic for approximate computing on encrypted numbers. To implement this, the input real numbers are first scaled with a large scaling factor and rounded to the nearest integer (quantization). Then they are encoded into plaintexts and subsequently encrypted into ciphertexts. To maintain a constant scaling factor in the ciphertext after multiplication, CKKS offers an efficient *rescaling* procedure which moves down the ciphertext to the next lower level by removing a prime factor from coefficient modulus q, at the same time scales down the amplified scaling factor in the ciphertext by the prime that removed from q. As mentioned before, one can drastically improve FHE performance via Single-Instruction Multiple-Data (SIMD) packing methods. In CKKS, a vector of up to $N/2$ complex numbers can be encoded in a single plaintext element. This allows one to perform parallelized SIMD homomorphic operations on packed ciphertexts efficiently. Packing can be viewed as if the ciphertext has independent slots, each concealing one data item. To manipulate the slots within a ciphertext, CKKS offers the *rotate* primitive that can circularly shift the data locations across the slots.

2.2 Neural Network Inference and Training

A feed-forward neural network composes of a stack of processing layers, where each layer performs certain computation on its input data according to the layer type, and outputs the processed data to the next layer for further computation. The common types of a nerual network layer are as follows.

- Convolution layer. This layer computes weighted sum of the input data. Each convolution operation is computing the dot product between a weight vector (i.e., filter map) and a data vector, and then adding a bias to it. The locations of the filter maps are shifted so as to compute with different data vectors from the whole input data.
- Fully connected layer. This layer can be viewed as a special kind of convolution layer, where the weighted sum (dot product) is always done between a weight vector and the whole input data.

– Activation layer. This layer applies an activation function to each of the input data. The activation functions are usually non-linear functions like Sigmoid, ReLU, etc.
– Pooling layers. This layer is usually used to down sample the input data to a smaller size, by returning the maximum (max-pooling) or average (average-pooling) of input vectors from the whole input data.

Neural networks are used for inference tasks like classification and regression. The inference phase only involves forward-propagation, where the input data is feed into the network, processed layer by layer, and the last layer gives the final output of the network. Before a neural network can be used for inference tasks, it must be trained. The training phase involves both forward-propagation and backward-propagation, whereas the backward-propagation is used to compute the derivatives (gradients) of a loss function with regard to the network weights and biases. An optimization algorithm (e.g., Gradient Descent) is then used to update the weights and biases according to the gradients, to minimize the value of the loss function.

A straightforward implementation of the Gradient Descent (GD) algorithm would be to update the weights and biases after each training sample, which is called Stochastic Gradient Descent (SGD). However, in practice people often adopt a more optimized form called Mini-Batch based GD, which accumulates the gradients from a batch of training samples, and then update the weights and biases at one time. Specifically, if the batch size equals the whole training set, the method is also called Batch Gradient Descent. In our secure training framework, we adopt Mini-Batch (Batch) Gradient Descent with HE packed data, to take full advantage of the performance benefit from SIMD-styled computations.

2.3 Transfer Learning

Deep learning (DL) is one of the most successful paradigms in data-driven approaches that has wide acclaimed performance in many practical applications. Yet, it works only under the assumption that training data and testing data are sampled from the same distribution, which may not hold at many practical scenarios. Naive approach is to train new model independently for each new data distribution. Training a new model from scratch for each new data distribution not only adds additional computational burdens but it also requires large amount of labeled data. Transfer Learning, which aims to transfer knowledge among different domains, can be a promising candidate to address the aforementioned challenges [24]. Different from DL, transfer learning leverages the knowledge from one or more source domains to maximize the performance in the target domain. Recently, transfer learning has been shown great capability with reducing the deep learning requirements for both computational requirements and the amount of labeled data [25]. Wide range of deep learning applications has benefited from transfer learning including Natural Language Processing (NLP), Computer Vision, and Robotics [26–28]. In our approach, we leverage transfer learning to realize efficient machine fault diagnosis across different operating conditions.

3 PrivGD: Secure Neural Network Training with FHE

In this section, we introduce our design and implementation of PrivGD, a FHE-based secure neural network training framework. We describe the components and considerations for a generate framework for different network architectures.

3.1 Matrix Multiplications with packed FHE ciphertexts

Matrix multiplication is a core operation in neural networks. To enable efficient matrix multiplications with encrypted data, we leverage the HE packing feature to pack multiple matrix elements into slots of a single ciphertext. This gives the dual benefits of reduced ciphertext amount and SIMD-styled parallel computation. Specifically, we adopt the following formats in PrivGD to pack a matrix into ciphertexts:

- Row-majored Packing (RP). A m-row n-column matrix $X^{m\times n}$ is packed into m ciphertexts, with Row $\boldsymbol{r}_i$ encrypted in ciphertext C_i $(1 \le i \le m)$.
- Column-majored Packing (CP). A Matrix $X^{m\times n}$ is packed into n ciphertexts, with Column $\boldsymbol{c}_j$ in Ciphertext C_j $(1 \le j \le n)$.
- Replicated Packing (REP). A Matrix $X^{m\times n}$ is packed into $m \times n$ ciphertexts, with each element $e_{i,j}$ is replicated in all the slots in Ciphertext $C_{i,j}$.

Subsequently, we define the following matrix multiplication operations.

- A REP matrix $X^{m\times k}$ multiplies a RP matrix $Y^{k\times n}$, the result $Z^{m\times n}$ is a RP matrix. In particular, we have the equation: $[C_i]_Z = \sum_{j=1}^{k}([C_{i,j}]_X \times [C_j]_Y)$, where $[C_i]_Z$ is the ciphertext for row $\boldsymbol{r}_i$ of Z, $[C_{i,j}]_X$ is the ciphertext for element $e_{i,j}$ of X, and $[C_j]_Y$ is the ciphertext for row $\boldsymbol{r}_j$ of Y.
- A CP matrix $X^{m\times k}$ multiplies a REP matrix $Y^{k\times n}$, the result $Z^{m\times n}$ is a CP matrix. Similarly, we have the equation: $[C_j]_Z = \sum_{i=1}^{k}([C_i]_X \times [C_{i,j}]_Y)$.
- A RP matrix $X^{m\times k}$ multiplies a CP matrix $Y^{k\times n}$, the result $Z^{m\times n}$ is a REP matrix. The ciphertext $[C_{i,j}]_Z$ for element $e_{i,j}$ of Z is produced by $[C_i]_X \times [C_j]_Y$, followed by applying the *AllSum* [29] algorithm to the multiplied ciphertext. Note that *AllSum* adds all the slots in a ciphertexts, and the sum is replicated in all the slots in that ciphertext. The algorithm uses $\log_2 N$ rotations and additions on the ciphertext, where N is the number of slots.

It should be noted that different matrix formats can be converted between each other by masking and rotation operations. For example, a REP matrix can be converted into a RP matrix, and the j-th slot of RP Matrix's i-th row $[C_i]_{RP}$ is produced by masking out (i.e., multiplying with a one-hot vector where only the masking location is one) the j-th slot of $[C_{i,j}]_{REP}$ in REP matrix and add into $[C_i]_{RP}$. However, matrix format conversions are generally expensive operations which cost additional multiplication depths and noise budgets, and should be avoided wherever possible.

In the next subsection, we will show how to utilize these matrix formats and multiplication functions to efficiently implement the neural network training processes.

3.2 Neural Network Training with FHE

Training a neural network generally involves multiple steps. For each iteration of training, first is to run the forward pass that takes in input and computes final output of the network, and second is to compute the loss function and its gradients with regard to the network output, and last is to run the backward pass that reversely computes the gradients for each layer using chain rule, and updates the weights and biases accordingly. To enable training with FHE, we target to solve the challenges in all these steps in PrivGD.

Forward Propagation. The linear computation layers, such as convolution and fully-connected layers, constitute the major computations in the forward pass. These layers are generally computing the weighted sum of layer inputs with regard to weights and biases, which can further be converted into matrix multiplications. As we adopt mini-batch based gradient descent algorithm in PrivGD, we target to compute the entire mini-batch in one shot through packed matrix multiplications described above. In particular, we adopt REP matrix format for weights and biases, and RP matrix format for batched inputs, where each column is an input vector and each row ciphertext pack a single element from each vector. The result of the matrix multiplication is another RP matrix that holds the output vectors, which is ready to be feed into the next layer for processing.

On the other hand, the non-linear activation layers, such as ReLU and Sigmoid, cannot be directly computed with FHE, and they need to be approximated by polynomials [2, 17], or implemented through private table lookups for a quantized version [30]. Max-pooling layers can be replaced with average-pooling or sum-pooling [2], which are actually special kinds of convolution layers with constant weights.

Loss and Gradient Computation. After getting the last linear-layer's output vector $\boldsymbol{z}$ from forward pass, we need to compute a loss function and its gradient with regard to $\boldsymbol{z}$. For classification tasks, $\boldsymbol{z}$ usually needs to go through another activation layer (Sigmoid or Softmax) before loss computation, and for regression tasks, it is usually directly used for loss computation. Table 3.2 summarizes the common loss functions and their gradients with regard to $\boldsymbol{z}$ for various tasks. For regression tasks, the gradients can be computed by the weighted difference between $\boldsymbol{z}$ and the ground truth label $\boldsymbol{t}$, which can be computed in HE directly. For classification tasks, the gradients can be computed by the difference between the last activation function outputs and the ground truth labels. While the Sigmoid activation function is easier to be approximated with polynomials and computed in FHE, the polynomial approximation for Softmax, however, is a harder problem and little work has been done on it to the best of our knowledge. In Subsection 3.3, we introduce a new and efficient way to train multi-class classifier in FHE with approximated Softmax.

Table 1. Common Loss functions and their Gradients.

Task	Activation funtion on $\boldsymbol{z}$	Loss function	Loss Gradient w.r.t $\boldsymbol{z}$
Binary classification	Sigmoid	Binary Cross-Entropy	Sigmoid($\boldsymbol{z}$) - $\boldsymbol{t}$
Multi-class classification	Softmax	Multi-class Cross-Entropy	Softmax($\boldsymbol{z}$) - $\boldsymbol{t}$
Regression	-	Mean Squared Error	$\frac{2}{N}(\boldsymbol{z} - \boldsymbol{t})$

Backward Propagation. After getting the gradients for the last linear-layer's outputs, we can start the backward pass and reversely compute the gradients for the weights and biases in all the layers. For each linear layer, two major types of computations are performed in the backward pass: one is to compute gradients for the layer weights and biases, and the other is to compute the gradients for the layer inputs (previous layers' outputs). Recall that the inputs X and outputs Y of each layer, as well as their gradients dX and dY, are stored as RP-formated matrices, and the layer weights and biases are stored as REP-formated matrices. According to chain rule, the gradient of each weight is the multiplication of the layer input and the gradient of the layer output it associates. Therefore, we have the following equation for gradient computation of layer weights.

$$[dW]_{REP} = [dY]_{RP} \times [X^T]_{CP} \tag{1}$$

In Equation 1, dY is the RP matrix holding layer output's gradients, and X^T is the transpose of layer inputs, which is in CP format. Their multiplication produces a REP matrix dW, in which each element is exactly the corresponding weight's gradient summed on the entire minibatch. For biases, their gradients db simply equals the gradients of the associated layer outputs, therefore can directly do a sum up for the minibatch using *AllSum* on the Ciphertexts. The summed gradients can be directly used to update the weights and biases in a later step. Pay attention that we do not take the additional step of computing the average gradients for the minibatch, as this can be combined with adjusting of the learning rate.

On the other hand, computing the gradients for the layer inputs, as shown in Equation 2, is very similar like the forward propagation process.

$$[dX]_{RP} = [W^T]_{REP} \times [dY]_{RP} \tag{2}$$

Back propagating through an activation function layer is different from a linear layer in two ways. First, there is no weights in the activation function layer, thus no weight gradients computation; second, the derivative of the activation function needs to be computed in order to compute the gradients of the layer input. As we use polynomials to approximate the activation functions (e.g., ReLU), we can take the derivative of the polynomial, which is also an polynomial, as the derivative of the activation function. On the other hand, for some activation functions the derivatives can also be computed in FHE in their native forms. For example, the derivative of Sigmoid layer $Y = Sigmoid(X)$ can be simply computed as $Y(1-Y)$, and the derivative of Tanh layer $Y = Tanh(X)$ is $1-Y^2$.

After the gradients of weights and biases in all the linear layers are computed, the next step is to update the weights and biases based on some optimization

method. The original version of SGD optimizer, $W = W - \eta \cdot dW$ where η is the learning rate, can be directly computed in FHE. One can also add a weight decay term or momentum term into the optimizer, but at the cost of some additional computational complexity.

3.3 Multi-Class Classifier Training in FHE with Approximated Softmax

Training a multi-class classifier in FHE requires approximating the Softmax function with polynomials, which is very challenging due to the fact that Softmax is a multi-variate function. In PrivGD, we do not target to directly approximate Softmax with polynomials. Instead, we approximate an estimated version of Softmax [31], which is proved to be able to achieve very close parameter estimations with original Softmax in multi-class classifier training. The output probability for each of the classes computed by the Estimated Softmax is described in Equation 3.

$$P_c = \prod_{m \neq c} Sigmoid(z_c - z_m) \tag{3}$$

Then, we can further compute the Negative Log Likelihood loss function and its gradients with regard to the last linear layer outputs, which can be expressed in Equation 4. We assume each training sample is encrypted and its class label is known to the training server, and thus it is straightforward to compute the gradients in Equation 4. In case the class labels are also encrypted, we just need some additional masking and addition operations for the gradient computation.

$$\begin{cases} dz_t = \sum_{m \neq t} (Sigmoid(z_t - z_m) - 1) & \text{for class t matches sample label} \\ dz_m = 1 - Sigmoid(z_t - z_m) & \text{for all the rest classes} \end{cases} \tag{4}$$

To compute the gradients in Equation 4 with FHE, we only need to approximate the Sigmoid function with polynomials, which is a simpler task as Sigmoid approximation has been widely studied in prior arts [32–34] and used in logistic regression training with FHE [17]. What's more, it should be noted that the multiplication depth for gradient computation in Equation 4 equals only a single Sigmoid approximation, which is the same as in logistic regression training.

3.4 Current Challenges and Our Approach

It should be noted that neural network training with FHE, although possible, still faces multiple challenges especially for larger networks: 1) deeper networks consume more multiplication depths as ciphertexts are computed throughout the layers, and 2) the multiplication depths are doubled in the training phase as it involves both forward and backward propagation; 3) the training losses usually need a large number of training iterations to converge, which further amplifies the multiplication depths; and 4) non-linear functions in the network may need to be

approximated with high-degree polynomials in order to be evaluated accurately in FHE. Therefore, in order to avoid the expensive bootstrapping operations, one needs very large encryption parameters to accommodate the large multiplication depths, which are deemed to be impractical due to high resource overhead and low performance.

Due to the above reasons, we are not targeting to train complete new models from scratch with FHE, instead, our approach is to use private data to refine existing models to make them adapt to new tasks, with a transfer learning approach. In later section, we will demonstrate the efficiency of our framework with a practical application to fine-tune machine fault diagnosis models with encrypted user data.

4 Secure Transfer Learning for Personalized Machine Fault Diagnosis

In this section, we demonstrate our new paradigm of private and personalized MLaaS through secure transfer learning, with the real-world application of machine fault diagnosis under different operating conditions.

4.1 The Machine Vibration Sensor Datasets

Our application scenario is to utilize deep learning models for diagnosing motor bearing faults from vibration sensor data attached to the machines. The datasets are downloaded from the Case Western Reserve University Bearing Data Center Website [35]. The CWRU bearing dataset is time-series data that collected at 12k sampling rate. It composes 4 different subsets which corresponds to different loading torques (i.e., operating conditions), where the torque values ranges from 0 to 3. In each subset, the data instances fall into 4 different categories, one normal category and three faulty categories including inner-race faults (IF), outer-race faults (OF), and bearing-race faults (BF). Each faulty category could have 3 fault sizes, i.e., 0.007 inches, 0.014 inches, and 0.021 inches, which leads to 10 total classes (1 normal class, and 9 faulty classes), as shown in Table 2.

4.2 Network Model for Machine Fault Diagnosis

Our model for the fault diagnosis is composed of two components, a feature extractor and a classifier. In particularly, the feature extractor is a 5-layer convolutional neural network with 1-dimensional kernels (1D-CNN). It aims to find a latent representation of the time-series data that could be class discriminative. On the other hand, the classifier which is composed of a fully connected layer followed by a Softmax activation layer, takes the extracted features from the 1D-CNN network as inputs, and outputs the probabilities the input sample belongs to each of the 10 classes. The detailed structure of our model is shown in Fig. 1.

Table 2. CWRU bearing dataset description [36]

Class Label	Fault Type	Fault Size (inches)	Load (hp)
1	Normal	0	0, 1, 2, 3
2	IF	0.007	0, 1, 2, 3
3	IF	0.014	0, 1, 2, 3
4	IF	0.021	0, 1, 2, 3
5	OF	0.007	0, 1, 2, 3
6	OF	0.014	0, 1, 2, 3
7	OF	0.021	0, 1, 2, 3
8	BF	0.007	0, 1, 2, 3
9	BF	0.014	0, 1, 2, 3
10	BF	0.021	0, 1, 2, 3

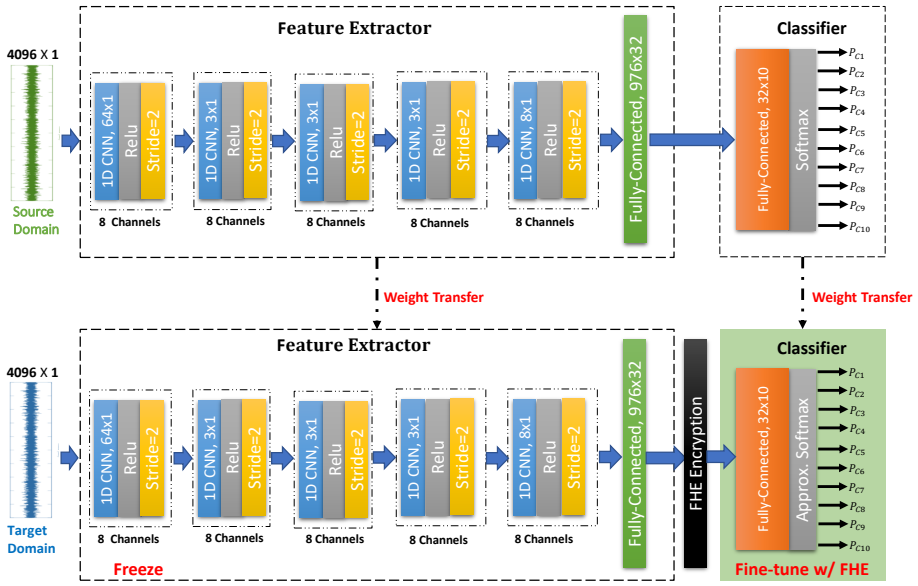

Fig. 1. Fault diagnosis model and the secure transfer learning approach.

4.3 Secure model fine-tuning across different operating conditions

In the MLaaS paradigm, the model owner deploys his model (i.e., source model) on a cloud server to provide inference services for other users. In our application scenario, we assume each user's machine is operating at a different condition, and directly applying the source model to the target conditions may lead to degraded diagnosis accuracy. To solve the problem, we propose a secure transfer learning approach, where a user can fine-tune the source model with encrypted data samples from his own machine and the corresponding working condition.

Specifically, the model owner first distributes the source model's feature extractor to all the users, and then the users use the feature extractor to extract features from their own data samples, encrypt the features with FHE, and send them to the cloud server to fine tune the classifier of the source model. It must be noted that, when the classifier is fine-tuned by a particular user, its weights and biases are becoming encrypted, and it can only be used to provide diagnosis services for that user after fine-tuning.

4.4 Implementation of Secure Fine-Tuning Process

We use PrivGD to implement the fine-tuning process. As shown in Fig. 1, the user utilizes the encrypted features as input to fine-tune the classifier part of the source model. PrivGD implements the Estimated Softmax for multi-class classifier training, and we only need to approximate Sigmoid for it to run in FHE. In [17], the authors suggested to use the Least Squared method to find polynomial approximations for Sigmoid on certain input interval. We adopt a similar approach and use the degree-3 polynomial $g(x) = 0.5 + 0.15012x - 0.00159x^3$ for approximating Sigmoid in our model. In order to minimize the number of training iterations in the fine-tuning process, we employ the batch gradient descent approach, in which each iteration uses all the training samples from the user. In our experiment setting, each user uses 2000 samples to fine-tune the source model, and each input feature dimension is 32, so the input for each training batch is a 32×2000 RP-formated matrix $[X^{32\times 2000}]_{RP}$ in ciphertexts.

Before starting the fine-tuning process, we need to fix the number of training epochs. The number needs to be carefully chosen in order to balance the required multiplication depth in FHE and the fine-tuning accuracy. For each training iteration, the following steps are involved: 1) the input features are first multiplied with the layer weights in the forward pass; 2) then the results are used to compute the loss gradients as with the formula described in Subsection 3.3 (multiplication-depth is 2 as we use degree-3 polynomial for Sigmoid); 3) and then the computed gradients are multiplied with the input features to get the gradients for the weights; 4) at last the weights are updated by subtracting the gradients multiplied with the learning rate. The total multiplication depth in one round of training iteration is 5. In our experiments, we will show that after fine-tuning the source model by 10 epochs, it already converges to optimal accuracy on the target data, therefore the total number of multiplicative depths in the whole fine-tuning process can be set to 50.

4.5 FHE Parameters selection

We choose CKKS to be the the underlying FHE scheme as it natively supports double-precision float-point numbers in neural network training. The CKKS scheme is governed by three major parameters, the ring dimension (polynomial modulus degree) N, the scaling factor Δ that controls the precision of the plaintext value, and the ciphertext coefficient modulus q that determines the largest multiplication depth D of a ciphertext. As with previous analysis, the

whole fine-tuning process needs a multiplication depth of 50, and this requires q to have at least 52 prime factors [3]. We select the first and the last factors to be 50-bit primes, and all the intermediate factors to be 30-bit primes. As a result, the ciphertext coefficient modulus q is to be 1600 bits in total. On the other hand, we select the scaling factor Δ to be 2^{30}, and the rescaling operation after each multiplication can maintain the same scaling factor for the plaintext value in the ciphertext. The last step is to choose an appropriate ring dimension N for the encryption scheme. On the one hand, we need a large enough N to meet the required security level, and on the other hand, we need to keep N as small as possible for more efficient FHE computation. Following the recommendation of NIST [37], we set the security level to be at least 80 bits, and according to the parameter estimation equation given in [17], we need N to be 2^{16}.

5 Experiment Evaluation

5.1 Experiment Server Setup

We carry out the experiments on a server with an Intel Xeon Platinum 8170 CPU @ 2.10 GHz with 26 cores, and 188 GB RAM. The operating system is Arch Linux. The fault diagnosis model training and fine-tuning on clear (i.e., unencrypted) data is done using Pytorch at version 1.3.1, and for the secure fine-tuning process we use our PrivGD framework implemented on Microsoft SEAL FHE library version 3.4.5.

5.2 Experiment Results

We have 4000 data samples in each of the four vibration sensor datasets as described in Section 4.1, denoted as 0hp, 1hp, 2hp, and 3hp according to the machine operating conditions. For each of the datasets, we first train a complete model (including feature extractor and classifier) with Pytorch and the original Softmax using all the 4000 samples on the clear data, in which 3000 randomly chosen samples are used as training set and the rest are used as test set. In each training, we vary the Mini-Batch size and learning rate to maximize the classification accuracy on the test set. The best test accuracy we can get are 97.6%, 97.75%, 98.15%, 98.65% on the four datasets respectively. Pay attention to the fact that these are the *same-domain* accuracy where the trained models are applied to the data from the same operating condition. In the following transfer learning experiments, the models will be in turn set as the source model, and be applied to the data from the other operating conditions, the test accuracy will be *cross-domain* accuracy.

For each transfer learning experiment, we select one operating condition as source domain, and the rest operating conditions as target domains. We separately

[3] For CKKS implementation in the SEAL library, the first prime is consumed in the encryption process, the last prime is used to accommodate the scaled plaintext value, and all the other primes in between are consumed one by one after each multiplication.

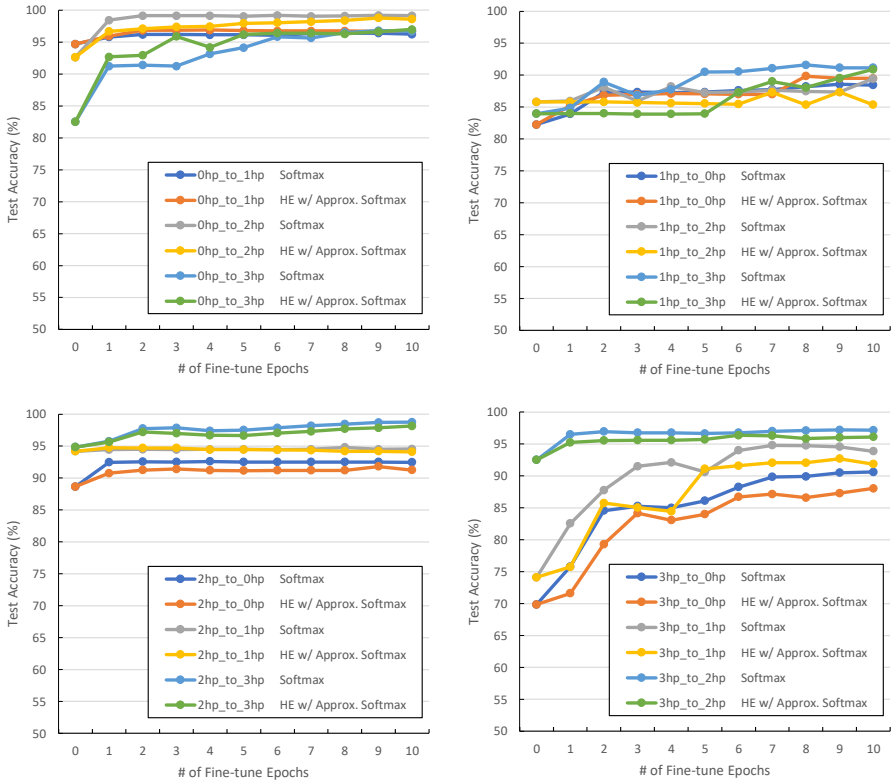

Fig. 2. Fine-tuned accuracy of transferred models on the target datasets.

fine-tune the source model on each of the target domains. For each fine-tuning process, we randomly choose 2000 samples from the target dataset and use Batch Gradient Descent to fine-tune the classifier of the source model, and another 1000 samples to test the accuracy of the fine-tuned model. Specifically, we implement two versions of the fine-tuning processes, one is the unencrypted version with Pytorch and the original Softmax function, and the other is the encrypted version with PrivGD and appproximated Softmax. We also log down the test accuracy after 1 to 10 fine-tuning epochs respectively, as shown in Fig. 2.

We can see that, before fine-tuning (i.e., at fine-tune epoch #0), the model accuracy generally drops quite significantly on the target data, compared with the *same-domain* accuracy. The fine-tuning process can efficiently improve the accuracy of the source model on the target data, especially in the first few epochs. The fine-tuning accuracy converges to the optimal after around 8 epochs, and after that, the improvement becomes marginal. The fine-tuning improvements are not the same for different experiments, with the maximal improvement of 20% across all the experiments (70% to 90% in the 3hp_to_0hp case). On the

other hand, the secure encrypted version achieves quite close accuracy to the unencrypted version, at most of the time the difference is within 3%.

5.3 Running Performances of Fine-tuning with FHE

Memory Usage. A CKKS ciphertext is composed of two degree-N polynomials, with q to be the polynomial coefficient modulus. As with the parameters chosen in our experiments, we can estimate that each ciphertext is 25MB. We can further compute that the total number of ciphertexts for the inputs, weights and biases, and outputs of the classifier as shown in Fig. 1 is 372. Therefore, the total memory resource usage is about 9.1 GB.

Latency Performance. The total run time for the 10 epochs of fine-tuning is 3.85 hours on our experiment server. Pay attention to the fact that the run time for each epoch gradually decreases with the number of epochs, where the first epoch takes the longest run time of 42.9 minutes, and the last epoch takes the shortest of 3.3 minutes. This is because the level of the ciphertexts is reduced by the *rescaling* operations along with the multiplication operations, which results in smaller coefficient modulus parameters and more efficient ciphertext operations.

6 Related Work

The major effort of our work is to apply transfer learning to the secure MLaaS scenario. In the convention of transfer learning, the feature extractor is usually considered as public and shared across different domains, while the domain-specific part of the model is trained or fine-tunned on domain-specific data. Several previous arts have demonstrated the applicability and efficiency of transfer learning with FHE-based secure MLaaS [3, 6]. For example, In [3], the authors proposed the workflow that the user used a public feature extractor to extract features from medical images, and then encrypted the features with FHE and sent to cloud for private inference. Similar like [3] and [6], in our approach, we assume the MLaaS service provider makes the feature extractor public to all the users, and puts the classifier on the cloud for private inference. However, our work further demonstrates that, if the classifier of the service provider was previously trained with data in a different domain (ie, working condition) from the user data, it may not work well on the user data. Therefore, what our approach is different from [3] and [6] is that, we propose to use a small amount of encrypted user data to fine-tune the original classifier of the service provider, and the fine-tuned classifier can provide higher inference accuracy on the user data. Our approach not only applies to the machine fault diagnosis task in our paper, but in fact provides a general paradigm that can be applied to other MLaaS tasks in similar use scenarios.

Our work belongs to the category of secure neural network training on encrypted data with FHE. Due to the large performance gap compared with the clear data counterpart, there are very limited prior arts in this category

and most of them focus on the simple task of logistic regression training [17, 38]. In [17], the authors tried to train a binary classifier on the encrypted medical images. They targeted to train a complete model from scratch, which needs many training iterations and subsequently very larger encryption parameters. On the contrast, we demonstrate a more practical way of applying secure training with transfer learning for the real world applications. We show that by fine-tuning on existing models, it requires much fewer training epochs and smaller encryption parameters, although for more complicated multi-class classification tasks.

Another category of related work is the secure inference of neural networks. CryptoNets [2] was the first to implement a inference network with FHE, but limited to the MNIST dataset. FasterCryptoNets [3] was among the first to try deeper networks and larger datasets with FHE, but suffered from high resources overhead. E2DM [5] and LoLa [6] tried to employ the SIMD packing feature to optimize the performance and resource overhead of inference network with FHE. These work commonly used polynomials to approximate the ReLU activation function inside the networks. As they didn't handle the training phase, there was no need to approximate the last Sigmoid or Softmax layers for the classification models. On the other hand, MPC-based solutions, such as Gazelle [13] and XONN [10], were free from approximation of non-linear functions in networks, but required both server and client to be constantly online and suffered from high communication overhead between them.

7 Conclusion

In this paper, we propose a new secure MLaaS paradigm, in which the user uses his private data to fine-tune the model on the cloud for higher inference accuracy. We build up PrivGD, a secure neural network training framework with FHE, and implement the fine-tuning process with it. In particular, PrivGD is the first to support the approximation of Softmax to train multi-class classifiers in FHE. We have demonstrated the efficiency of our secure transfer learning approach on the machine fault diagnosis tasks and datasets. In the future, we plan to apply our framework and approach to more real-world tasks and datasets.

References

1. Peter Peduzzi, John Concato, Elizabeth Kemper, Theodore R Holford, and Alvan R Feinstein. A simulation study of the number of events per variable in logistic regression analysis. *Journal of clinical epidemiology*, 49(12):1373–1379, 1996.
2. Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.
3. Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

4. Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. The alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. *arXiv preprint arXiv:1811.00778*, 2018.
5. Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222, 2018.
6. Alon Brutzkus, Oren Elisha, and Ran Gilad-Bachrach. Low latency privacy preserving inference. *arXiv preprint arXiv:1812.10659*, 2018.
7. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.
8. Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.
9. Chao Jin, Ahmad Al Badawi, Balagopal Unnikrishnan, Jie Lin, Chan Fook Mun, James M Brown, J Peter Campbell, Michael Chiang, Jayashree Kalpathy-Cramer, Vijay Ramaseshan Chandrasekhar, et al. Carenets: compact and resource-efficient cnn for homomorphic inference on encrypted medical images. *arXiv preprint arXiv:1901.10074*, 2019.
10. M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. {XONN}: Xnor-based oblivious deep neural network inference. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1501–1518, 2019.
11. Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.
12. Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.
13. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.
14. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.
15. Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.
16. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229, 1987.
17. Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2):e19, 2018.
18. Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.
19. Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

20. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.
21. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.
22. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.
23. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.
24. Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.
25. Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.
26. Jianfei Yu and Jing Jiang. Learning sentence embeddings with auxiliary tasks for cross-domain sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 236–246, 2016.
27. Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.
28. Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, pages 262–270, 2017.
29. Shai Halevi and Victor Shoup. Algorithms in helib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.
30. Patricia Thaine, Sergey Gorbunov, and Gerald Penn. Efficient evaluation of activation functions over encrypted data. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 57–63. IEEE, 2019.
31. Michalis Titsias RC AUEB. One-vs-each approximation to softmax for scalable estimation of probabilities. In *Advances in Neural Information Processing Systems*, pages 4161–4169, 2016.
32. K Basterretxea, Jose Manuel Tarela, and I Del Campo. Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons. *IEE Proceedings-Circuits, Devices and Systems*, 151(1):18–24, 2004.
33. Miroslav Vlcek. Chebyshev polynomial approximation for activation sigmoid function. *Neural Network World*, 4(12):387–393, 2012.
34. Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.
35. Case Western Reserve University Bearing Data Center. Motor bearing fault datasets. https://csegroups.case.edu/bearingdatacenter/home.
36. Guo-Qian Jiang, Ping Xie, Xiao Wang, Meng Chen, and Qun He. Intelligent fault diagnosis of rotary machinery based on unsupervised multiscale representation learning. *Chinese Journal of Mechanical Engineering*, 30(6):1314–1324, 2017.
37. Elaine Barker, William Barker, William Burr, William Polk, Miles Smid, et al. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.

38. Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):83, 2018.