International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Achieving Highest Privacy Preservation Using Efficient Machine Learning Technique

Pinkal Jain*[1], Dr. Vikas Thada[2], Shailesh Kumar Vyas[3]

Submitted: 20/01/2024 **Revised**: 15/03/2024 **Accepted**: 20/03/2024

Abstract: Data privacy has become a paramount concern in big data, prompting the development of encryption algorithms and security strategies to safeguard sensitive information. Centralized machine learning approaches often involve transferring data to a central point to train models, which poses a risk of data exposure because unauthorized persons can disclose our private data publicly. To address this issue, multi-party privacy protection combined with machine learning offers a solution, with machine learning emerging as a way to ensure privacy in multi-party settings. This paper presents EPFML (Efficient Privacy Framework Using Machine Learning) that employs data modification. The algorithm enables joint model training while maintaining multi-party security. We use gradient descent with encrypted data transmission, preventing data exposure during the process. To counter member inference attacks, we employ data modification on the data, ensuring data privacy. Our approach demonstrates applicability across various domains, offering a privacy-protected multi-party machine learning framework. Experimental results indicate the efficiency and accuracy of our method, paving the way for enhanced data security and privacy in multi-party learning environments.

Keywords: *Privacy, Privacy Preservation, Data Modification, Machine Learning, Gradient descent, Multi Party Privacy.*

1. Introduction

One of the biggest and most important concerns nowadays is data privacy, particularly in the big data era. There are several encryption algorithms and security strategies available which aim to protect sensitive information. Additionally, several security strategies allow only users with keys to access the data. This is made possible with the help of centralized learning, wherein data is collected, transferred to a central checkpoint to train a model. The process of data transfer faces the risk of the sensitive data being exposed to hackers. Therefore, preventing data exposure during the data transfer process is an important issue to achieve data security.

An approach to ensuring data privacy is to combine multi-party privacy protection [1] with machine learning where several users share their data and jointly learn from the pooled data while maintaining the security of their own information. This is possible through federated learning which can address data privacy issues in a multi-party environment. This paper, presents EPFML, a machine learning privacy-based technique using homomorphic encryption. We jointly trained the model using gradient learning while maintaining multi-party security. In each iteration round, we optimized the model using gradient descent ensuring that each user could learn from other users' information through transmission of the gradient. One concern was the possibility of member inference attacks wherein during the model training process, hackers could train their own shadow models using the plaintext gradient thereby compromising data privacy. In order to avoid this, we employed homomorphic encryption processes on the data, where users could perform calculations on the encrypted data. We found that upon decryption of the data, the result was similar to operations on the plaintext data thereby ensuring the quality and efficiency of the process. As encrypted data was used during the entire process, data privacy and data security was ensured.

The machine learning algorithm and data modification process proposed in this paper has several practical applications. Proposed work provides privacy-protected framework using machine learning to achieve data privacy. Furthermore, it can ensure data privacy in a multi-party learning environment. We tested our model using MNIST and metal-fatigue strength datasets, and calculated accuracy rate, time taken for the homomorphic encryption, and impact of various network structures and key lengths.
Section 2 presents a literature review that formed the foundation of our work. In section 3, we have provided an overview of the Improved Paillier federated network algorithm in terms of network structure, interaction, and security. The outcomes of the experiment are shown in Section 4, and the paper's summary is given in Section 5.

2. Related Work

2.1. Distributed machine learning

It's a type of multiple-node ML technique which aims to enhance accuracy, performance, and easily scale the data. Distributed ML environment [2] aimed to address the issues with ordinary synchronization involving training a huge model comprising a large volume of data by means of a state synchronous parallel model. Another framework aimed to systematically solve data and model parallel changes in a larger scale. A factor broadcast

[1] *Amity University Gwalior, MP-474020, India*
ORCID ID: https://orcid.org/0000-0001-8002-320X
[2] *Amity University Gwalior, MP-474020, India*
ORCID ID:
[3] *G H Raisoni University, Saikheda, MP-480106, India*
ORCID ID: https://orcid.org/0009-0003-1262-7353X
** Corresponding Author Email: pinku029jain@gmail.com*

calculation model has also been proposed which enables distributed learning of a parameterized large matrix model. Network communication's [3] efficiency under a particular bandwidth was improved so that parallel errors could be reduced and data parallel large-scale applications could be theoretically fused.

Google released Disbelief in 2012 [3], which is a capable of dividing a framework into 32 different nodes for the purpose of performing calculations. The InfiniBand network was released in 2013 through which model parallelism and distributed learning was introduced in deep learning. The efficiency of training of distributed Gradient Descent [4] in data parallel and model was compared theoretically and it was found that increasing the size also increases the efficiency.

2.2 Homomorphic encryption and secure multi-party computation

In distributed machine learning, a central network assigns tasks to external users and so, the data is accessible by all users and there is no data privacy in the system. Multi-party computing [5] is usually involved in distributed learning where unknown or complicated computing processes are revealed to third parties. One of the first methods to be proposed was the Garbred circuit method which was used to solve general and simple problems such as two-party password issues. Several years later, SMPC (Secure Multi-Party Computation) was introduced. Currently, SMPC represents a sub-category of cryptography that allows distributed users to collaborate in computing functions while maintaining the privacy of their information.

Homomorphic encryption has slowly gained popularity in the recent years. It was initially proposed to be used for bank applications in 1978. It has been used to create multiplicative homomorphism in one of the first cryptosystems, RSA (Rivest-Shamir Adleman) [6]. The Paillier algorithm [7] was developed in 1999. This algorithm combined with homomorphism has been used in applications concerning digital auction, retrieval of cloud ciphertext, and digital elections among others. In 2009, an algorithm known as fully or complete homomorphic encryption, or FHE, was proposed, which is based on lattices that adhered to the rules of additive as well as multiplicative homomorphism. FHE has been applied in several cases owing to its high security. It has proven useful especially in the field of cloud computing [8] where it has ensured data privacy.

Another technique to maintain data privacy is differential privacy which adds noise to the data and thereby prevents data exposure. However, if noise is introduced when the data size is less, it can impact model's accuracy.

2.3 Machine learning

Initially, federated learning was used for updating the models for Android users locally. Later in2019, researchers at Google employed Tensorflow [9] to create a scalable manufacturing system for multi-party collaborative learning on mobile devices. Furthermore, in the same year, the problem of adjusting the parameters of learning models when data was divided among multiple nodes, while ensuring that the raw data was not sent to a central network. A framework known as Secure Boost [10] has been proposed which has an accuracy equivalent to that of a total of five privacy protection technique.
Several applications of federated learning [11] have been documented. Google designed the Gboard system which can recognize keyboard input, carry out predictions, protect privacy, and enhance input efficiency of users [12] . In the healthcare field, machine learning has been used to protect sensitive medical information of patients. The processing of natural language and recommendation systems are two other uses of federated learning.

Over the recent years, considerable efforts have been dedicated to harnessing the capabilities of machine learning algorithms for the purpose of enhancing privacy protection. Differential privacy has been used for this purpose, and SMC [13] has been used to address the noise that arises from the use of differential privacy. A batch crypt algorithm has been proposed which was developed by optimizing the FATE framework [14]. This algorithm encodes a long integer in the place of a group of quantized gradients, and carries out encryption of a single gradient at a time, thereby increasing the encryption and decryption efficiency, and decreasing the calculation required.

3. Proposed Modelling

3.1 Generation of an algorithm using federated learning

In data areolation, there are several intermediate variables that interact with each other during the training phase. At this point, it is possible to use optimize model (Figure 3.1). This forms the basis for federated learning. Federated learning is of two types: sample expansions or horizontal, and feature expansions or vertical learning.

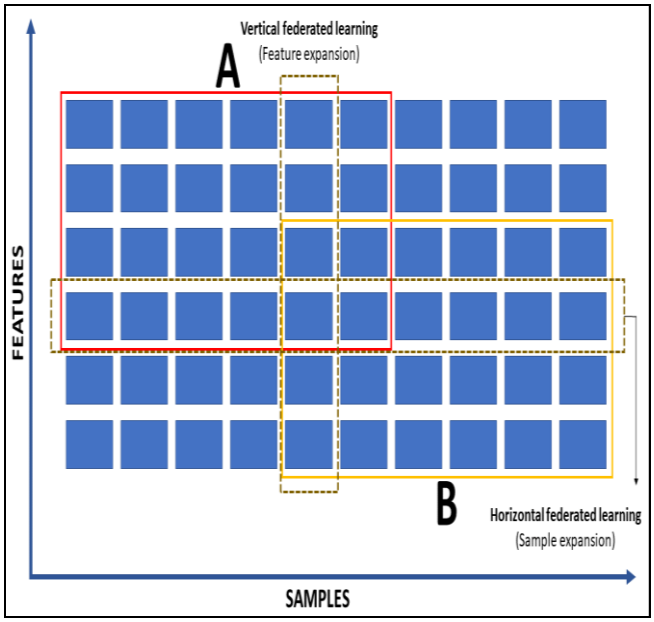

Figure 3.1: Horizontal and vertical federated learning indicated by dotted row and column respectively. 'A' and 'B' are the two data owners.

Horizontal federated learning encompasses machine learning. Considering that the D stands for data, X stands for features, I stands for data index and Y stands for samples and the equation for horizontal federated learning can be represented as:

$$X_i = X_j, Y_i = Y_j, I_i \neq I_j, \forall D_i, D_j, i \neq j$$

Based on this equation, there may or may not be intersections between various users that have different data. Horizontal

federated learning aims to enable different users to pool their data to train a model, while also guaranteeing the privacy of sensitive information. For this to take place, all users' data need to be aligned to ensure that all the users contribute to similar framework and the iterations that take place in the model are synchronous. In contrast, for vertical federated learning, the data of all users involved in the training have different features.

3.2 Federated network algorithm

The federated learning network proposed here aims to enable all users to train model during the training phase. Assuming that most of the neural networks undergo training by gradient descent, gradients have been chosen as intermediate variables here. Even though the data cannot be represented directly by the gradients, It is possible to depict the connection between the model's structure with the data, which can help in training the model. Figure 3.2 shows the federated learning network's architecture which comprises of various learning clients and a computing server.

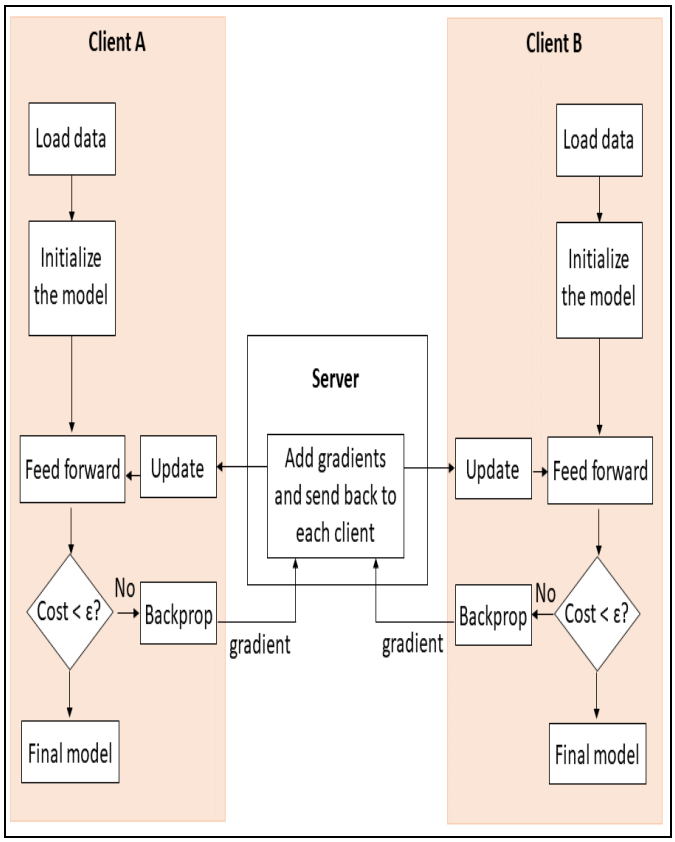

Figure 3.2: Federated learning based neural network architecture

3.2.1 Learning client

Learning clients have their own data, the quantitative dimensions of which are aligned with other users' data before training the model. The main functions of the learning client are to perform extraction of gradients during training, computation of the gradients using the computing server, collection of server responses, passing of the results, making updates to the model, and performing repeated iterations for the convergence of the model.

3.2.2 Computing server

A platform that is in between throughout the process of learning is represented by the computational server. The server's primary duties include gathering gradient data from users, conducting calculations using the gradients, integration of data obtained from multiple models, and transmission of the result to each individual user.

3.2.3Federated multi-layer perceptron algorithm

In this paper, classic multiple-layer perceptron algorithm served as the foundation for the development of the federated multiple-layer perceptron (FMLP) algorithm. Another name for this kind of technique is a deep feed-forward network and belongs to the category of deep learning models. FMLP algorithm is capable of training simple models for individual clients in an environment of multi-party data areolation by means of a gradient-sharing process. An FMLP network model shown in figure 3.3.

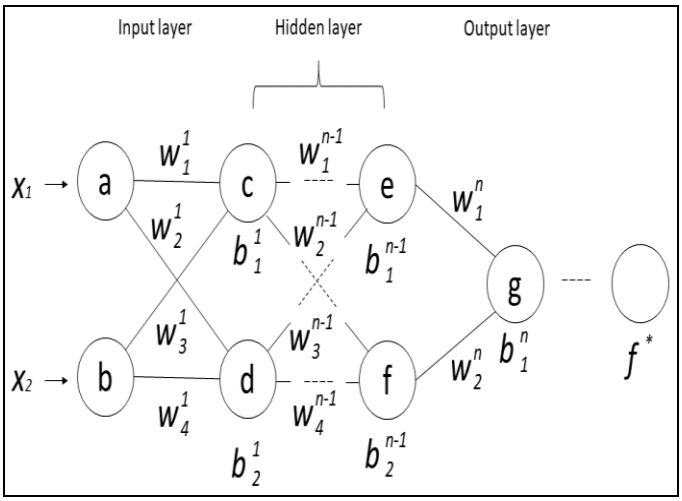

Figure 3.3: Multi-layer perceptron model

Given the model parameter θ, represented by {w1.....wn, b1.....bn}, and the learning rate during training denoted as lr, the dataset is denoted by x, comprising {x1.....xn}. The primary goal of the model is to approximate the distribution f*. The forward process of the network involves computing the training output through the following formula:

$$out = fp(x, \theta)$$

The formula for loss function is as follows:

$$c = loss(f^*(x), out)$$

The back-propagation calculates gradients then propagates backward so that parameters can be adjusted accordingly and the error can be decreased. The back-propagation can be calculated using the following formula:

$$grad = bp(x, \theta, c)$$

The model undergoes updates by modifying the network parameters in accordance with the gradients acquired following the back-propagation process, which is expressed using the following formula:

$$\theta' = \theta - lr \cdot grad$$

Therefore, when a multi-layer perceptron (MLP) recognizes a federated network that, in turn, is determined by the target output. The recent gradient data is fused and the gradient descent is accelerated by the model during the learning process with the help of server. After integrating the variation in gradient

information provided by every user, the server computes a new gradient and sends it to every user so that the model is updated. Finally, model convergence takes place when each user's loss is less than ε, and the same federated model is sent to all users. Table 1 shows steps involved in the functioning of the FMLP algorithm.

Table 1: FML perceptron
Input: Consider a Dataset x
Output: Model θ final
1: Firstly, we initialize the parameters
2: In the cycle, each n does
3: Forward propagation: outn = fp(xn, θn);
4: Calculate loss: cn = loss(f*(xn),outn);
5: if cn < ε then
6: Break
7: else
8: Calculate gradn = bp(xn,θn,cn);
9: After that the gradients value send to server then the computing server will provide new value of gradients;
10: Modify: θn+1 = θn – lr * new_gradient;
11: close if and for loop
12: return θ

3.2.4 Paillier federated network

Several individuals can use specific information to do collaborative machine learning utilizing the FMLP method introduced in this work. However, hackers not only require the data of each user but also the final updated model.
Evidence from a member inference attack shows that hackers can gain access to the computing server (Shokri et al., 2017). Using ensemble learning, hackers can derive a predicted model from these shadow models that is similar to the actual trained model. Therefore, the federated algorithm only addresses the problem of data security.
To address model security, federated learning can be added with other techniques. In homomorphic encryption, plaintext (a) is encrypted to ciphertext (c), by performing few operations and results in encryption a plaintext. The overall encryption process is represented using the following formula:

$$E(a) \oplus E(b) = E(a \otimes b)$$

Here, E stands for the encryption algorithm, a and b stand for different plaintexts, and ⊕ and ⊗ are the operators.
Homomorphic encryption performs its function based on the operator. For instance, when a multiplication operator is present, then the multiplicative homomorphism is satisfied, an example of which is the RSA algorithm (Calderbank, 2007). On the other hand, if an addition operator is present, the additive homomorphism is satisfied, and an example of this is the Paillier algorithm (Paillier, 1999). Furthermore, if both the multiplication and the addition operators are present, then both the homomorphisms are satisfied (Gentry, 2009). In our FMLP algorithm, the gradient data from all users are summed together, and so, the
Paillier algorithm can perform the additive homomorphic encryption.

3.2.4.1 Pailler algorithm

Key generation: Select two primes p and q such that the values are large, are of equal length, and satisfy the formula of $(p * q, (p-1) * (q-1)) = 1$. Then, n and λ can be calculated using the following formulas:

$$n = p \cdot q$$
$$\lambda = lcm(p-1, q-1)$$

Then, select value g which satisfies the formula, $g \in Z^*_{n^2}$, such that the order of g can be divided by n. The following equations can then be used to calculate μ:

$$\mu = (L(g^{\lambda} \bmod n^2))^{-1} \bmod n$$

Encryption: Considering m to be the plaintext and c to be the ciphertext. The following formula can be used to represent the method of encryption utilizing the public key:

$$c = g^m \cdot r^n \bmod n^2$$

Decryption: Similarly, the decryption process using the private key can be denoted using the following formula:

$$m = L(c^{\lambda} \bmod n^2) \cdot \mu \bmod n$$

3.2.4.2 Improved Paillier algorithm

The complexity that occurs in the Paillier method throughout the steps of encryption and decryption. Hence, we have used an updated version of the Paillier algorithm whose accuracy and efficiency have been previously reported (Jost et al., 2015). The three steps of this improved algorithm are given below:

Key generation: Considering α to be the divisor, the order of g in the public key can be represented as αn.
Encryption: Considering r to be a random value, m to be plaintext and c to be ciphertext as follows:

$$c = g^m \cdot (g^n)^r \bmod n^2$$

Decryption: The decryption process can be represented as follows:

$$m = \frac{L(c^{\alpha} \bmod n^2)}{L(g^{\alpha} \bmod n^2)} \bmod n$$

The strength of the improved Paillier algorithm can be seen in the decryption equation where α is used instead of λ. In this equation, the power operations number changes from 2·λ to 2·α, thereby significantly reducing the overhead time as α is a divisor of λ. The enhanced algorithm's complexity in computation may be expressed as O(|n|2 |α|), whereas the conventional method's operational complexity being O(|n|3).

3.3 Paillier federated network architecture

Paillier encryption is used for protection of the gradients thereby ensuring that even if the computing server is attacked, the specific data from each gradient are not accessible by the hacker. Furthermore, hackers cannot make use of the encrypted gradient data for training of the shadow models.

In general, Paillier encryption needs key pairs for performing its function; therefore, a key management centre (KMC) is added to the algorithm for the generation and management of key pairs. Thus, the model comprises of the computing server, learning clients, and KMC (Figure 4).

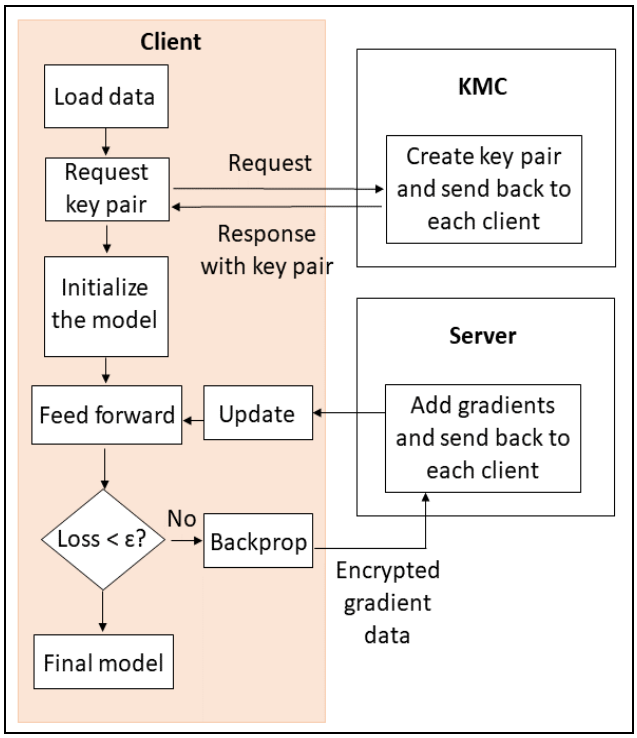

Figure 3.4: Paillier federated network architecture

EPFMLP follows the FMLP concept, with the addition of KMC, which is introduced since the instructional client needs to send an inquiry to KMC prior to learning. Key pairs are created by the KMC and returns them to the users after verifying that every user is online. Next, using the encrypted data, each user performs multi-party machine learning. The algorithm for the learning client in EPFMLP is provided in Table 2. The three extra processes carried out by EP-FMLP involve encryption and decryption, homomorphic operations, and creation and distribution of key pairs.

Table 2: Algorithm for EPFMLP in the learning client

1. Begin by acquiring key pairs from the Key Management Center (KMC).
2. Initialize parameters.
3. Iterate through the dataset:
   a. Conduct forward propagation: compute outn using the function fp(xn, θn).
   b. Evaluate the loss: determine ci as the loss between the true labels f*(xn) and the model output outn.
   c. If cn is below a specified threshold ε, exit the loop.
   d. Otherwise, perform backward propagation: calculate gradn using bp(xn, θn, cn).
   e. Encrypt
Enc(gradn) = EncPailli-er(Publickey, gradn).
   f. Transmit the data (Enc(gradn)) to the server and receive the updated encrypted gradient (Enc(gradinew)).
   g. Decrypt the gradient using the private key of client n: grann = DecPailli-er(Privatekey, Enc(gradn)).
   h. Update the model parameters: θn+1 = θn – learning rate (lr) multiplied by gradnew.
4. End the iteration loop.
5. Return the final model with parameters θ final.

The local model is not updated immediately after the calculation of the gradients by the learning client. The gradient data is homomorphically encrypted and transmitted to the computing server, following which the learning client waits for performing operations. Once the learning client receives and decrypts the gradient data, the model is updated. This ensures privacy of other users' data. The algorithm for the KMC in EPFMLP is given in Table 3.

Table 3: Algorithm for EPFMLP in KMC

1. Continuously monitor incoming requests from clients.
2. Upon receiving a request:
   a. Create a fresh KeyPair.
   b. Provide the generated KeyPair to the requesting learning client.
3. Continue listening for requests in an ongoing loop.

On demand, the computational server delivers the gradient data returned by the computer's learning client after homomorphically synthesizing the encrypted data that it received from the client. Table 4 provides the algorithm for the compute server in EPFMLP. In this case too, data privacy is ensured during model training as the generated key pairs are not transmitted to the computing server.

Table 4: Algorithm for EPFMLP in the computing server

1. Continuously monitor incoming requests from clients.
2. Set up and initialize a container for gradient data, denoted as GradientData.
3. Upon receiving a request:
   a. Add the received encrypted data.
   b. If total number of received requests equals the number of learning clients:
      i. Iterate through the learning clients:
         - Aggregate the encrypted data for each client and append it to the GradientData.
      ii. Send the aggregated GradientData back to each respective client.
      iii. Terminate the process.
4. Continue listening for requests in an ongoing loop.

Security analysis of the algorithm

The KMC cannot access the gradient data, nor can it access the data that has been encrypted by the learning client using the key, therefore it cannot be a part of security breach. The learning client encrypts the ciphertext. To perform operations, the computer server does not need to decipher the data. Because of this, all of the data on the server itself remains secret, meaning that hackers cannot access it regardless of whether they manage to breach the server.

Following receipt of the key pairing by the KMC, a learning client transmits the data that has been encrypted to the server that's it analyses and returns the result with encryption to the learning clients. This also ensures that a user cannot access the data of other users. If hackers gain access to the computing server, they can only access the ciphertext form of the data. Even if the hackers can gain access to some of the intermediate results sent by the server, key pairs can be altered in each iteration process and so, hackers cannot access the final result sent by the server.

4. RESULTS AND DISCUSSIONS

In this section all the results and the discussions should be made.

4.1 Datasets and environment

Verification of the algorithm was performed using. Furthermore, 784 input layers are present in the neural network model along with two hidden layers.

On the other hand, the metal fatigue strength dataset comprises of 437 records that are derived from the NIMS. MatNavi is a materials database which encompasses alloys, ceramics, polymers, composites, superconducting materials, and diffusion samples. In this study, 437 samples were selected from MatNavi and a regression model was generated for testing different metals such as carburizing steel based on characteristics of the rolled product and conditions of heat treatment.

There are 15 dimensional features and a single dimensional label for each sample. The entire dataset is divided into four categories (Table 5). Overall, the model that was used in our study comprised of a single input layer comprising of 15 dimensions, a single output layer comprising of four units, and three hidden layers comprising of 64 units.

Table 5: Metal strength dataset for tasks of classification

Dataset_Name	Range	Count
Fatigue	[200, 400]	56
	[400, 500]	147
	[500, 600]	148
	[600, ∞]	86

Table 6: EPFMLP network structure

Dataset_Name	Input (Layer)	Hidden (Layer)	Output (Layer)
MNIST	784	2(layers) 64	10 (units)
Fatigue	16	3(layers) 64	4 (units)

The value of $D_n^{Dataset}$ denotes the value of the nth sample of the dataset. The experiments performed to test the EPFMLP algorithm and its optimization included assessing the prediction accuracy of the FMLP and single-node MLP, time used in training the model with varying key lengths, time used in training the model using varying sizes of the units of hidden layers, and influence of learning user count on the model's performance.

The environment used for the experiment was Windows 10, and Python. A computing server, several learning clients, and a KMC were deployed from the local area network, and Socket was used to establish communication between the various machines. The layout of the network that was used in the study is given in Figure 3.5.

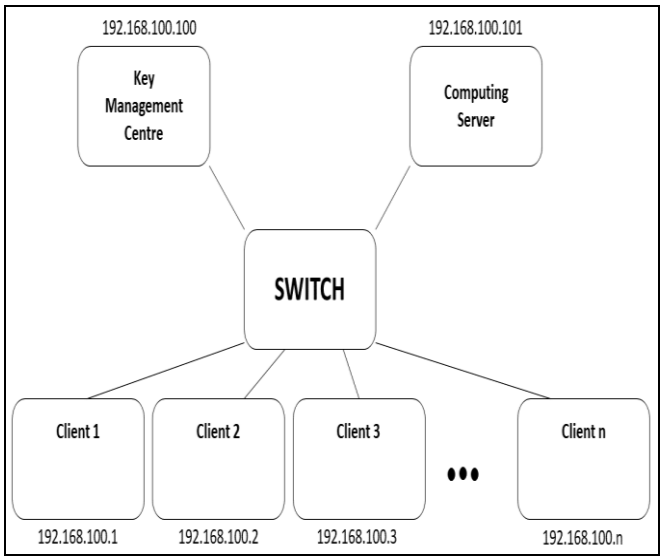

Figure 3.5: Network deployment

4.2 Comparison of prediction accuracy

Comparison between the EPFMLP and different algorithms was carried out while using the same dataset and network structure during training of the model.

For example, for the MNIST dataset, the first 4000 samples were selected as the training data (Dmnist), which were divided into two parts: $D_I^{mnist} = \{D_1^{mnist}, \dots, D_{2000}^{mnist}\}$ and $D_{II}^{mnist} = \{D_{2001}^{mnist}, \dots, D_{4000}^{mnist}\}$. For the test data, 10,000 samples from the MNIST dataset were used.
Similarly, for the metal fatigue strength dataset, 400 samples were selected (DFatigue) and were randomized [DFatigue' = random (DFatigue)]. This was divided into two subsets:

$D_I^{Fatigue'} = \{D_1^{Fatigue'}, \dots, D_{200}^{Fatigue'}\}$
and $D_{II}^{Fatigue'} = \{D_{201}^{Fatigue'}, \dots, D_{400}^{Fatigue'}\}$.

Overall, 70% of the data was used for training and 30% was used as test dataset. The results of the experiment are given in Table 7.
Table 7: Comparison table

Dataset_Name	Data_Subset	Algorithm	Accuracy
MNIST	D_1^{mnist}	MLP	0.8333
	D_2^{mnist}	MLP	0.9033
	Dmnist	MLP	0.9245
	D_1^{mnist}	EPFMLP	0.9252
	D_2^{mnist}	EPFMLP	0.9252
Fatigue	$D_1^{Fatigue'}$	MLP	0.9013
	$D_2^{Fatigue'}$	MLP	0.7833
	DFatigue'	MLP	0.8583
	$D_1^{Fatigue'}$	EPFMLP	0.8833
	$D_2^{Fatigue'}$	EPFMLP	0.8167
	DFatigue'	EPFMLP	0.8500

Based on results, accuracy of the model is higher in case of EPFMLP compared. The model that is trained by EPFMLP is similar to model trained by MLP. An accuracy rate of 0.9252 was reached, while an accuracy of 0.9245 was reached for the MLP model, with a difference of just 0.007 between the two models. In case of the metal fatigue strength dataset, a weighted average of both experimental results was performed as the same EPFMLP trained each client's model. The prediction accuracy for the MLP model was 0.858, with a difference of 0.008 between the two

models. Therefore, our findings indicate that both the EPFMLP and MLP algorithms can train models with similar prediction accuracies.

4.3 Comparison of time for model training

Network is vulnerable to attacks by hackers who may use the data to train their shadow models. This not only breaches the model security of the client to whom the data belongs, but also places other clients' data at risk. Therefore, in order to encrypt the data, we used the Paillier homomorphic encryption in EPFMLP. This encryption was operated during the transmission of data, and all homomorphic operations were carried out in the server to tighten the security of the data.

Optimal functioning of the Paillier encryption method is the key length. Higher security is provided by longer keys in general. However, the problem with using longer keys is that it increases

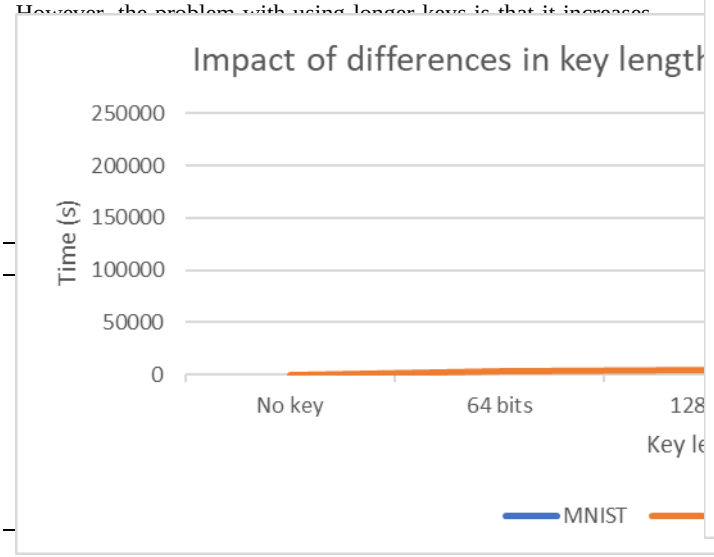

Figure 3.6: Impact of differences in key lengths.

Our findings indicate that in Paillier encryption, the length of the key directly impacts the time used for training the model. It follows that with an increase in the key length, the time taken for training the model will also increase. Therefore, an appropriate length of the key can help achieve a balance between data security and time consumption. Furthermore, if we want to enhance the security.

From our experimental run, using the same key length and the same model, 358.14 s is used for 4000 pieces of data, 733.69 s is used for 8000 pieces of data, and 1284.06 s is used for 12,000 pieces of data for every iteration. Therefore, the amount of data directly impacts the time used for model training. Furthermore, the iterations to compare the time taken for data encryption and decryption (Table 9 and Figure 3.7). It is evident from our results that the improved Paillier algorithm showed a 25 to 28% better performance compared to the native Paillier algorithm.

Table 9: Impact of differences in key lengths

Method	Key length (bits)	Time (s)
Paillier	128	1068.38
	256	6411.23
	512	35,930.83
Improved Paillier	128	779.37
	256	4716.37
	512	26,148.56

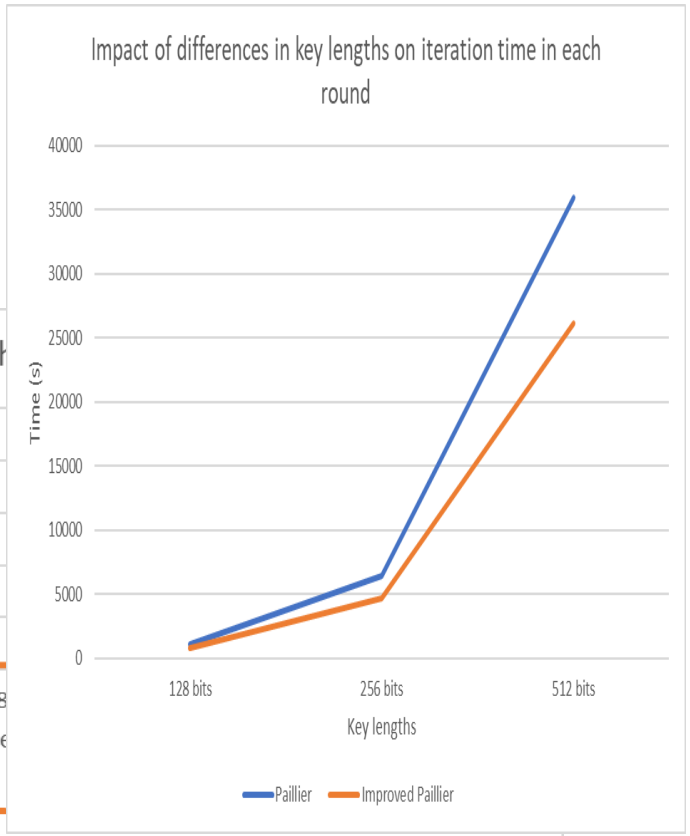

Figure 3.7: Impact of differences in model training for the Paillier and improved Paillier algorithms.

4.4 Comparison

The time and performance for both the forward and backward propagation is affected by the size of each layer. We tested this relationship through our experimental runs on the two datasets (Table 10 and Figure 3.8).

Table 10: Impact of differences in key lengths.

Dataset	Size of hidden layer (units)	Time (s)
MNIST	2 . 64	12,033.25
	2 . 128	23,981.02
	2 . 256	47,702.87
Fatigue	3 . 64	2615.42
	3 . 128	6941.04
	3 . 256	21,782.07

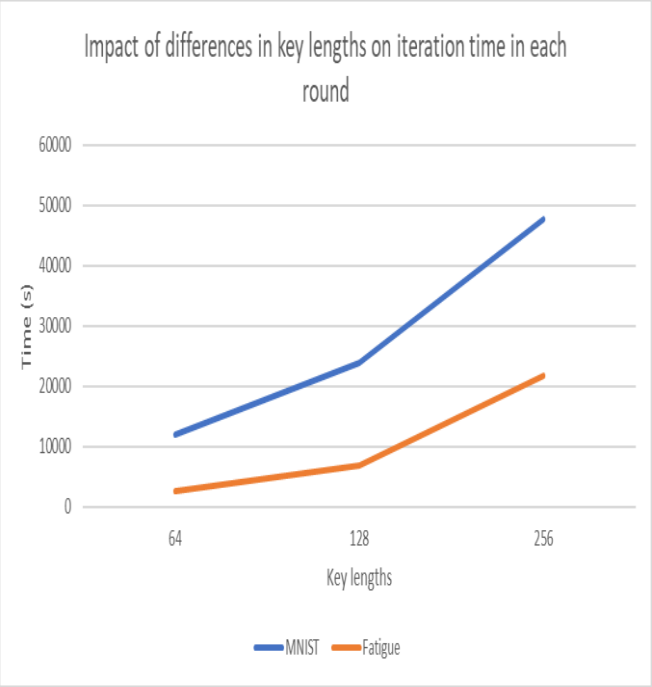

Figure 3.8: Impact of differences in key lengths of model training for the MNIST and fatigue datasets.

There is also an increase in the data that is transmitted over the network. Therefore, if we want to reduce the time, we should also reduce the layers.

4.5 Comparison of number of learning clients

Multi-party machine learning is possible through the use of EPFMLP algorithm. Theoretically, with increase in the number of learning clients.

Table 11: Accuracy of EPFMLP on fatigue dataset with various learning clients

Method	Dataset	Local_Accuracy	Logical_Accuracy
MLP	$D_{0-400}^{Fatigue}$	0.858	-
MLP	$D_{0-200}^{Fatigue}$	0.833	-
MLP	$D_{200-400}^{Fatigue}$	0.783	-
2-Client-EPFMLP	$D_{0-200}^{Fatigue}$	0.867	0.850
2-Client-EPFMLP	$D_{200-400}^{Fatigue}$	0.833	0.850
MLP	$D_{0-100}^{Fatigue}$	0.767	-
MLP	$D_{100-200}^{Fatigue}$	0.933	-
MLP	$D_{200-300}^{Fatigue}$	0.800	-
MLP	$D_{300-400}^{Fatigue}$	0.600	-
4-Client-EPFMLP	$D_{0-100}^{Fatigue}$	0.833	0.850
4-Client-EPFMLP	$D_{100-200}^{Fatigue}$	0.967	0.850
4-Client-EPFMLP	$D_{200-300}^{Fatigue}$	0.867	0.850
4-Client-EPFMLP	$D_{300-400}^{Fatigue}$	0.733	0.850

The prediction accuracy was significantly improved when using the multi-client EPFMLP algorithm. However, the logical accuracy was similar when using two or four clients. The local accuracy for the EPFMLP algorithm was also significantly improved and was amplified in extreme cases, for instance, in the 4-Client run, the last client had outlier data despite which EPFMLP is having higher accuracy than the accuracy of MLP. The local accuracy for the second client was 93.3% which was improved by 3.4% when using EPFMLP. Rather, it is similar to the model performance in case of MLP. As the model performance depends on batch expansion, if each client transmits lesser amount of data, then the learning per round will also reduce thereby reducing the time.

5. CONCLUSION & FUTURE WORK

In this research, we offer a multi-party confidentiality-protected learning algorithm that uses data modification-based confidentiality and machine learning to ensure that various individuals are able to utilize it without jeopardizing the safety and confidentiality of their personal information. When considering data security, our proposed algorithm can train models in multiple rounds in a way that is similar to the model being trained in a single round using all data together. All users transmit their respective data over the network and data modification is performed on the data over the server. The new data thus obtained forms the basis for updating the model. However, data modification affects the model performance by increasing the workload involved in encryption and decryption, which in turn affects the efficiency of training the model. Also, the final model performance is also affected by the structure of the network, length of the key, and replacement frequency of the key.

In future, machine learning can be made more efficient and scalable by using vertical federated learning algorithms which can divide the features among the users. Secondly, the model performance should be improved by increasing the efficiency of the encryption process. Finally, other privacy-protected learning algorithms can be considered that are more robust such as anti-malicious attack client algorithms and hybrid algorithms.

Author contributions

Pinkal Jain1: Conceptualization, Methodology, Software, Field study
Dr. Vikas Thada2: Data curation, Writing-Original draft preparation, Software, Validation., Field study
Shailesh Kumar Vyas3: Visualization, Investigation, Writing-Reviewing and Editing.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Ning, Z., Dong, P., Wang, X., Hu, X., Guo, L., Hu, B., Guo, Y., Qiu, T., & Kwok, R. (2020). Mobile edge computing enabled 5g health monitoring for internet of medical things: A decentralized game theoretic approach. IEEE Journal on Selected Areas in Communications, 39, 463–478.

[2] Liao, H., Zhou, Z., Zhao, X., Zhang, L., Mumtaz, S., Jolfaei, A.,

Ahmed, S. H., & Bashir, A. K. (2020). Learning-based context-aware resource allocation for edge-computing-empowered industrial IoT. IEEE Internet of Things Journal, 7(5), 4260–4277.

[3] Petropoulos, A., Sikeridis, D., & Antonakopoulos, T. (2020). Wearable smart health advisors: An imuenabled posture monitor. IEEE Consumer Electronics Magazine.

[4] Rahmani, A. M., Gia, T. N., Negash, B., Anzanpour, A., Azimi, I., Jiang, M., & Liljeberg, P. (2018). Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach. Future Generation Computer Systems, 78, 641–658.

[5] Tuli, S., Tuli, S., Wander, G., Wander, P., Gill, S. S., Dustdar, S., et al. (2020). Next generation technologies for smart healthcare: challenges, vision, model, trends and future directions. Internet Technology Letters, 3(2), e145.

[6] Ud Din, I., Guizani, M., Hassan, S., Kim, B., Khurram Khan, M., Atiquzzaman, M., & Ahmed, S. H. (2019). The internet of things: A review of enabled technologies and future challenges. IEEE Access, 7, 7606–7640.

[7] Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, R., et al. (2022). AI for next generation computing: Emerging trends and future directions. Internet of Things, 19, 100514.

[8] Tariq, N., Asim, M., Al-Obeidat, F., Zubair Farooqi, M., Baker, T., Hammoudeh, M., & Ghafr, I. (2019). The security of big data in fog-enabled IoT applications including blockchain: a survey. Sensors, 19(8), 1788.

[9] Moqurrab, S. A., Anjum, A., Manzoor, U., Nefti, S., Ahmad, N., & Ur Rehman Malik, S. (2017). Differential average diversity: An efcient privacy mechanism for electronic health records. Journal of Medical Imaging and Health Informatics, 7(6), 1177–1187.

[10] Li, J., Sun, A., Han, J., & Li, C. (2020). A survey on deep learning for named entity recognition. IEEE Transactions on Knowledge and Data Engineering

[11] Moqurrab, A., Ayub, U., Anjum, A., Asghar, S., & Srivastava, G. (2021). An accurate deep learning model for clinical entity recognition from clinical notes. IEEE Journal of Biomedical and Health Informatics.

[12] Ma, J., Huang, X., Mu, Y., & Deng, R. H. (2020). Authenticated data redaction with accountability and transparency. IEEE Transactions on Dependable and Secure Computing.

[13] Tariq, N., Khan, F. A., & Asim, M. (2021). Security challenges and requirements for smart internet of things applications: A comprehensive analysis. Procedia Computer Science, 191, 425–430.

[14] Tariq, N., Asim, M., Khan, F. A., Baker, T., Khalid, U., & Derhab, A. (2021). A blockchain-based multi-mobile code-driven trust mechanism for detecting internal attacks in internet of things. Sensors, 21(1), 23.

[15] Shukla, S., Thakur, S., Hussain, S., Breslin, J. G., & Jameel, S. M. (2021). Identification and authentication in healthcare internet-of-things using integrated fog computing based blockchain model. Internet of Things, 15, 100422.

[16] Buyya, R. H., Calheiros, R. N., & Dastjerdi, A. V. (2016). Big Data: Principles and Paradigms. Morgan Kaufmann.

[17] Iwendi, C., Moqurrab, S. A., Anjum, A., Khan, S., Mohan, S., & Srivastava, G. (2020). N-sanitization: A semantic privacy-preserving framework for unstructured medical datasets. Computer Communications.

[18] Habibi, M., Weber, L., Neves, M., Wiegandt, D. L., & Leser, U. (2017). Deep learning with word embeddings improves biomedical named entity recognition. Bioinformatics, 33(14), i37–i48.

[19] Unanue, I. J., Borzeshi, E. Z., & Piccardi, M. (2017). Recurrent neural networks with specialized word embeddings for health-domain named-entity recognition. Journal of Biomedical Informatics, 76, 102–109.

[20] Zhu, H., Paschalidis, I. C., & Tahmasebi, A. (2018). Clinical concept extraction with contextual word embedding. arXiv preprint arXiv:1810.10566.

[21] Si, Y., Wang, J., Xu, H., & Roberts, K. (2019). Enhancing clinical concept extraction with contextual embeddings. Journal of the American Medical Informatics Association, 26(11), 1297–1304.

[22] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). Biobert: A pre-trained biomedical language representation model for biomedical text mining. Bioinformatics, 36(4), 1234–1240.

[23] Batet, M., & Sánchez, D. (2014). Privacy protection of textual medical documents. In: IEEE Network Operations and Management Symposium (NOMS). IEEE, pp. 1–6.

[24] Sanchez, D., & Batet, M. (2017). Toward sensitive document release with privacy guarantees. Engineering Applications of Artificial Intelligence, 59, 23–34.

[25] Batet, M., & Sánchez, D. (2019). Leveraging synonymy and polysemy to improve semantic similarity assessments based on intrinsic information content. Artificial Intelligence Review, 53, 2023–2041.