Secure Model Fusion for Distributed Learning Using Partial Homomorphic Encryption

Changchang Liu(✉), Supriyo Chakraborty, and Dinesh Verma

Distributed AI Department, IBM T J Watson Research Center,
Yorktown Heights, NY 10598, USA
Changchang.Liu33@ibm.com, {supriyo,dverma}@us.ibm.com

Abstract. Distributed learning has emerged as a useful tool for analyzing data stored in multiple geographic locations, especially when the distributed data sets are large and hard to move around, or the data owner is reluctant to put data into the Cloud due to privacy concerns. In distributed learning, only the locally computed models are uploaded to the fusion server, which however may still cause privacy issues since the fusion server could implement various inference attacks from its observations. To address this problem, we propose a secure distributed learning system that aims to utilize the *additive* property of partial homomorphic encryption to prevent direct exposure of the computed models to the fusion server. Furthermore, we propose two optimization mechanisms for applying partial homomorphic encryption to model parameters in order to improve the overall efficiency. Through experimental analysis, we demonstrate the effectiveness of our proposed mechanisms in practical distributed learning systems. Furthermore, we analyze the relationship between the computational time in the training process and several important system parameters, which can serve as a useful guide for selecting proper parameters for balancing the trade-off among model accuracy, model security and system overhead.

1 Introduction

Today, massive amounts of data are distributed across multiple geographic locations. To mine the data for useful analytics, the commonly used approach is to collect all the data in a centralized location and train an AI model on the aggregated data. However, there are many situations where this approach may not be feasible. In particular, sharing data across national or organizational boundaries may not always be permitted.

In order to address these situations, approaches for federated learning [1–5] have been proposed. Federated learning enables models to be built at different locations and then brought together by a fusion operation to create a single model that captures the patterns present at all the different locations. Many flavors of federated learning can be used, depending on the nature of the relationship that exists between different sites hosting the distributed data.

S. Calo et al. (Eds.): PADG 2018, LNCS 11550, pp. 154–179, 2019.
https://doi.org/10.1007/978-3-030-17277-0_9

Federated learning shares some methodologies with distributed machine learning [6–8] which aims to handle complicated models computed over big training datasets. However, the main difference is that federated learning needs to deal with a system where different participating sites are not able to communicate over a high bandwidth, low latency and high reliability network, a typical situation when going across multiple organizations.

Since federated learning typically deals with the training of models across multiple organizations, its basic approach is to share model parameters instead of sharing the training data itself. By migrating models instead of migrating data for training, it can integrate the artificial intelligence insights from data found in different systems and locations.

While existing approaches for federated learning tend to focus on performance metrics such as the time required to train a model optimally over bandwidth-constrained networks [4] or achieving model accuracy metrics comparable to centralized learning [1,5], a key aspect of federated learning across multiple organizations is that of trust. Current work assumes complete trust among different sites, and that each site is comfortable sharing model parameters with each other. However, the organizations involved may not completely trust each other.

Trust and regulatory compliance is one of the motivating drivers for federated learning. By sharing only model parameters instead of training data, the privacy of individual data elements can be protected. However, even when local training data is not directly exposed to build models, the models and even the local training data are still susceptible to various inference attacks [9,10]. This leads to a need to protect and encrypt model parameters when building models. In this paper, we present an approach for federated learning which enables model building without exchanging model parameters in the clear.

Our approach uses partial homomorphic encryption to enable federated learning in an environment where model movement outside of the location can not be done in the clear. We also provide two optimization mechanisms to mitigate the performance hit that arises in distributed applications with the use of encryption schemes by: (1) constructing encryption/decryption lookup tables based on model parameter quantization; and, (2) leveraging batch encryption.

By enabling distributed fusion in this manner, we are increasing the autonomy of the distributed systems. Specifically, we are providing schemes that can address some of the privacy and data protection constraints that arise due to organizational policies, and provide a mechanism to enable learning in the presence of such restrictive policies.

We begin this paper with a discussion of the high-level federated learning architecture and the use-cases that motivate the use of partial homomorphic encryption. This is followed by a description of our approach, a discussion of issues associated with the performance aspects of using encryption for federated learning, followed by evaluation of some performance optimization mechanisms. We also provide an overview of related work, and conclude with a discussion of the limitations of our technique and open areas of investigation.

2 Architecture and Use Cases

We assume that federated learning happens in the environment shown in Fig. 1. There are multiple sites, each site with its share of training data, and an ability to create a neural network model from such data. Each site has an additional component, the federation agent, which is responsible for working with a federation server that performs model fusion. The federation agent takes the model resulting from the training process, provides it to the federation server, retrieves the fused model from the federation server, and interfaces with the learning mechanism present locally.

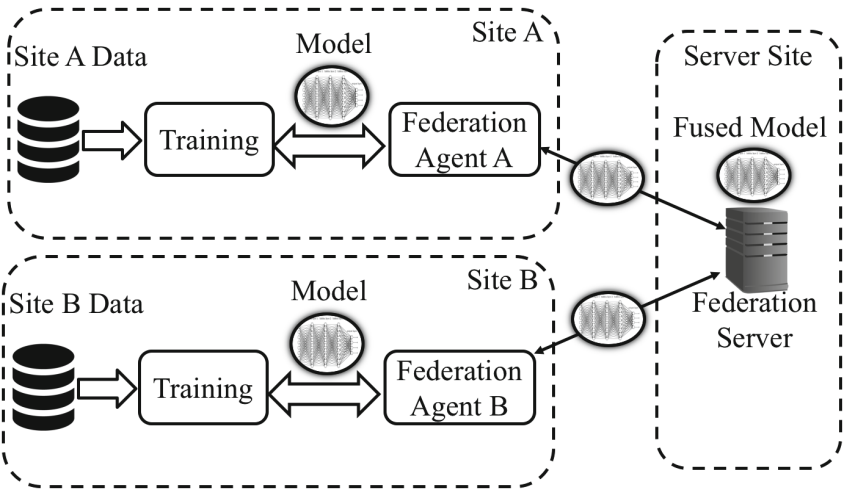


Fig. 1. Machine learning with data from partners

When the federation agents and federation server trust each other, the approach for federated learning is addressed by the algorithms and papers published previously [1–5]. Those algorithms exchange model parameters with each other in the clear to attain the goal of a common model trained over all of the data. However, there are many situations where the agents and the server may be running at sites which may not completely trust each other. In those cases, the model parameters can not be shared in the clear. In the following subsections, we discuss some of the common use-cases in which the exchange of model parameters may be problematic.

Depending on the trust relationship among different agents and servers, the system may be divided into one or more trust zones. Entities belonging to the same trust zone can share data and/or model parameters with each other. However, entities belonging to different trust zones should not get access to the raw data or the model parameters in the clear when the model parameters deal with the results of training performed on data in another trust zone.

2.1 Cloud Based Federation

One case of federated learning consists of the situation where the Federation Server is run and operated as a cloud-hosted service, while the data is present within different branches of a company. As an example, a bank may have its local data warehouses at three different locations, and would like to build a common AI model across the three data warehouses. It can leverage the federation server hosted at a cloud provider site, but may not be willing or able to send either the local data or the model parameters resulting from training a local model to the federation server in the cloud.

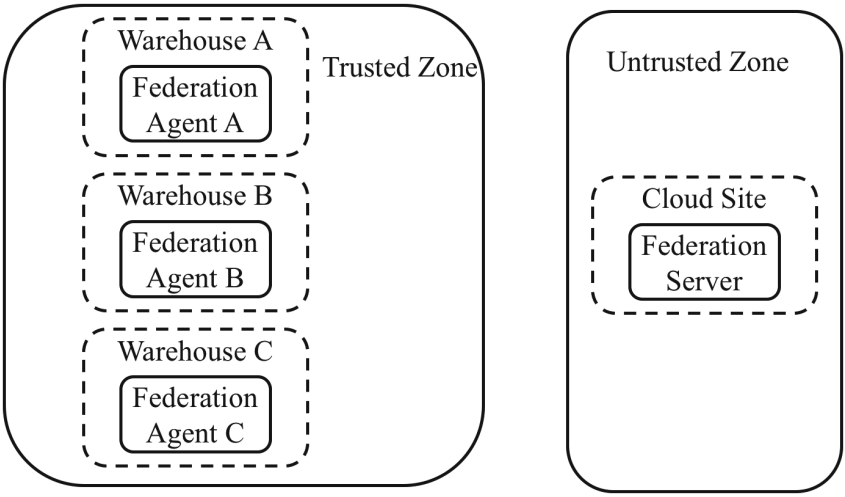


Fig. 2. Cloud based federation scenario

The situation is shown in Fig. 2. The trust relationship among the different federation agents, which operate in environments belonging to the same organization, is that of complete trust. However, there is limited trust between the agents and the federation server. The agents can not trust the federation server with the model parameters that are being sent to it.

In this case, one can argue that the federation server should be set up in a trust zone which can be trusted by the agents, e.g., a private Federation Server owned by the financial company. However, setting up a private service is a slow and time-consuming process, whereas using the ready-made services on an active already-running federation service in the cloud could accelerate the task of model building substantially.

This challenge of cloud based federation can arise in any industry which is subject to strict regulatory restrictions, including but not limited to financial companies, health-care companies, public sector companies, and military organizations.

2.2 Consortium and Alliances

In many industries, consortia are formed to enable sharing of knowledge and information among different organizations. As an example, it is not uncommon for health-care organizations of different countries to form an alliance to share information about health-care data with each other. Several joint organizations have been formed to share information about agriculture, social issues, and demographics among member organizations. While some information can be shared with each other, portions of the data may contain sensitive details which different consortia members may not feel comfortable sharing with each other. In these cases, sharing of models among the alliance members may be permitted, or even enabled by a service belonging to the alliance, but the alliance members may not want the other partners to see the model parameters that each individual member contributes. Furthermore, each member organization may not want to have the server see its parameters in the clear either.

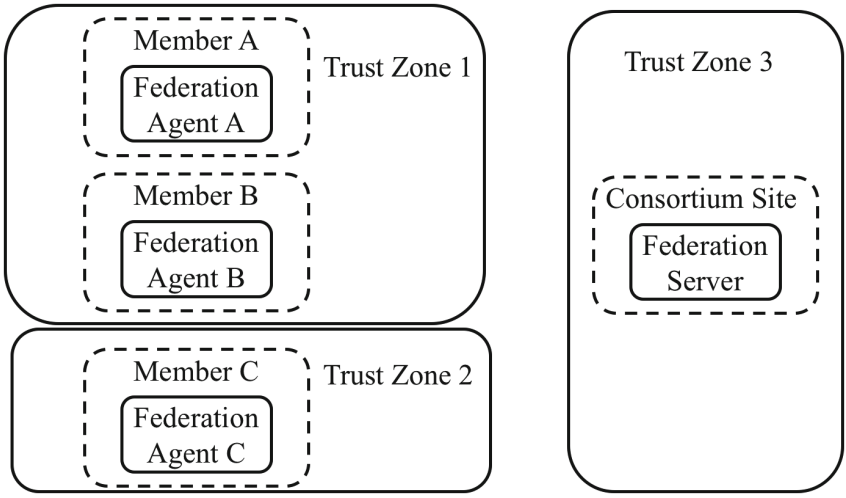


Fig. 3. Consortium scenario

In this scenario, each of the alliance members is in its own separate trust zone. Some of the alliance members may trust each other, but not all members are trusted equally. Such a trust zone relationship is shown in Fig. 3, where two of the agents belong to the same trust zone, while a third agent and the federation server belong to separate trust zones.

2.3 Military Coalitions

Modern military operations are frequently conducted within coalitions, where more than one country joins forces to conduct missions in a collaborative manner. Such coalition operations are a common occurrence in peace-keeping, where

multiple countries work together to maintain peace in a region subject to conflict. They are also used for humanitarian operations in cases of natural disasters.

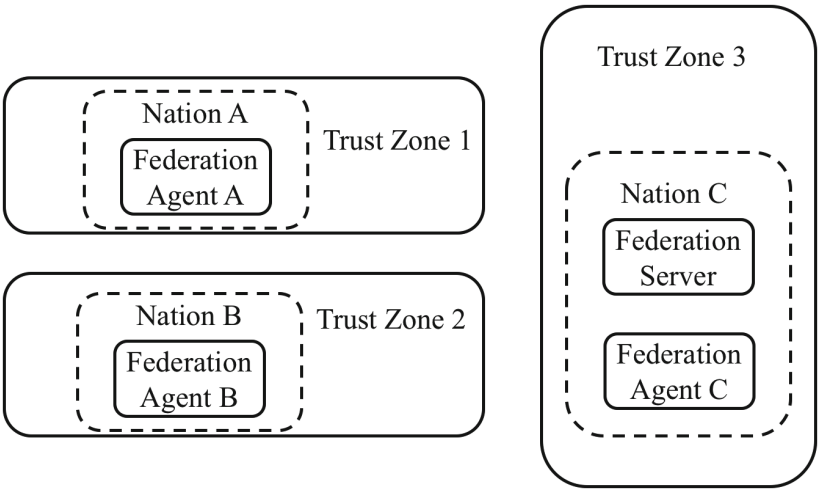


Fig. 4. Military coalition scenario

As coalitions work together, almost every member nation collects data from its operations. Such data may consist of video footage from surveillance, audio recordings and seismic readings from sensors, or tabular data that is maintained manually by the personnel involved in the joint effort. Much of this data can be shared among coalition members to improve their operations. As an example, the video footage of insurgents collected by the member nations can be shared to improve the AI models used to detect insurgency and to alert the peace-keepers.

Coalition members may not be able to share raw data, but may be able to share model parameters with each other. One specific case in which that may arise in shown in Fig. 4, where one of the coalition members is training its model but using the information available from all of the coalition partners. Each coalition partner would be willing to share the models, but not if the model parameters are sent in the clear. The trust zone for federation server includes the federation agent that belongs to the same nation, but each of the other agents are in their own trust zones.

A situation analogous to coalition operations can also arise in other emergency situations where different government agencies are cooperating together in any joint operation, e.g., reacting to a natural disaster, or planning for a special event. Different types of regulations may prevent complete sharing of raw data or unprotected model parameters among the agencies, but they would be able to train models and share the models with each other, especially if the model parameters can be encrypted during the fusion process.

We assume that in each of the above scenarios, the model parameters cannot be shared in the clear among entities that are not in the same trust zone. Thus, two federation agents in different trust zones should not be able to see the individual model parameters that are trained from any of the agents in the other trust zones. If the federation agent and the federation server are in different trust zones, the federation server should not be able to see the model parameters provided by the federation agent.

3 Our Proposed Mechanism

Although there are many different methods to perform distributed learning, we present our method for secure fusion in terms of a synchronized online approach described in papers such as [1] and [4]. While these form only one of the many flavors of distributed learning that are possible, as enumerated in [3], the approach proposed in this paper can be extended to other flavors as well.

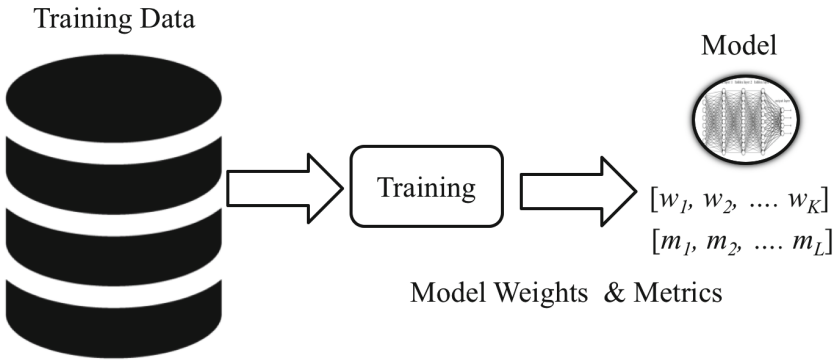


Fig. 5. Abstract representation of machine learning

We focus on the task of training a neural network for classification to illustrate our approach to distributed learning. The typical process of supervised machine learning for a neural network takes as an input training data, and tries to determine the weights on the neural network which can best estimate the patterns that are present within the training data. Although there are many different types of machine learning algorithms, and many different types of neural networks with different interconnections among different nodes, we can model the machine learning process as a black box which takes as an input the training data, and outputs a list of numeric weights which best represent the function/pattern present in the training data. The simplified model of training the machine learning model is shown in Fig. 5. The training process produces some K weights $[w_1, w_2, \ldots w_K]$ from the input data. Additionally, it might also produce some L metrics $[m_1, m_2, \ldots m_L]$ which may represent attributes such as the

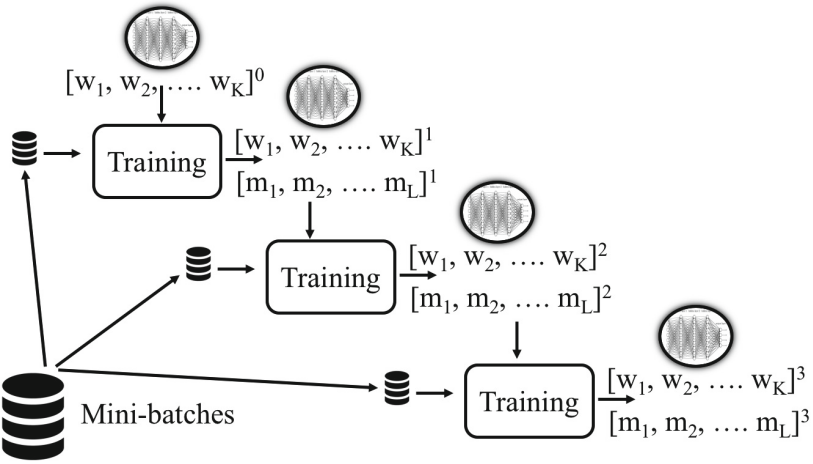


Fig. 6. Machine learning with mini-batches

accuracy of the model, the loss metric associated with the training process, or other metrics such as the count of the number of samples in the training data.

In neural network training, the training process usually takes the training data and divides it into several mini-batches as illustrated in Fig. 6. The big data set shown in Fig. 6 on the lower left side is divided into multiple smaller data sets, or mini-batches. The machine learning process is usually initiated with a random set of weights, and the error in prediction over the mini-batch estimated using a network with the specified weights. The error is then used to determine how to adjust the weights to come up with a new set of weights that can reduce this error. The process is then repeated with the next mini-batch. These cycles are repeated until all the data is examined one or more times, and the weights eventually converge. This mechanism is leveraged during the federated learning process.

3.1 Federated Learning Model

We first present the approach for federated learning described in [1,2] and [4], which all follow the mini-batch oriented process outlined in Fig. 6. In federated learning, the approach is to have each of the different sites start with a neural network with the same set of weights that are initialized randomly. The Federation Server may provide the agents with the initial set of weights.

Each mini-batch training step at each agent is modified somewhat. Once the agent finishes calculating the new weights over a mini-batch, it sends these new weights and associated metrics to the federation server. The federation server then computes a function over these weights. The function could be an average across the different weights provided by the agents, or it could be a weighted average where different priorities are given to different sites, and the priorities

are used as coefficients to compute the weighted average (i.e., the priorities are used as weights for averaging). We use the term priority to reduce confusion with the elements being sent which are the weights on the neural network.

The process is shown in Fig. 7 with three participating sites. Each of the sites computes its weights and metrics locally and sends it to the federation server. The federation server combines the weights, and sends the resulting fused model back to each of the sites. The sites can then proceed with the training process for the next mini-batch. We consider a distributed system with N agents, each with access to a local dataset consisting of labeled samples, i.e., the i-th agent has access to a local data set d_i. The pseudo-code for the training process is given below:

- 1: Each agent contacts the fusion server to get hyper-parameters for training. Note that each agent will train the same type of model (such as a neural network).
- 2: Each agent trains its model on a local mini-batch of the data. For instance, each model can be characterized by a weight vector.
- 3: Each agent sends the weights to the fusion server, which computes a function of the weight values provided by the agents. This function could be an average of the values supplied by the agents. In other cases, agents may be given different priorities in the computation of the average. If agent i has a priority p_i, the function could be $\hat{\boldsymbol{w}} = \sum_i p_i \boldsymbol{w}_i$.
- The steps 2 through 3 are repeated until convergence.

In some cases, the function f shown in Fig. 7 would be simple averaging, while in other cases it may be averaging with priorities at the end of each mini-batch. In some other cases, e.g., using flavor 2 of federated learning as described in [3], the function f would be the identity function, and the federation server sends the results of the function computation back to the participating nodes using a random sequence.

Our main challenge is to enable the federation server to continue to perform these functions while not being able to see the model parameters in the clear. The agents at each site would be sending the model parameters to the federation server after encryption.

3.2 Secure Model Fusion

It has been shown [1,4] that the above process leads to the same model at the fused location as would be produced by using all the data from the distributed sites to train a model at a single site, for loss functions that are additive, which are the most common types of loss functions that are used. However, this federated learning mechanism still faces the challenge of revealing the weights of the model to the federation server which may not be trusted.

To counter this situation, we can use fully homomorphic encryption or FHE [11] to completely perform the fusion for the encrypted local models at the fusion server. However, FHE causes a significant degradation in performance, with each

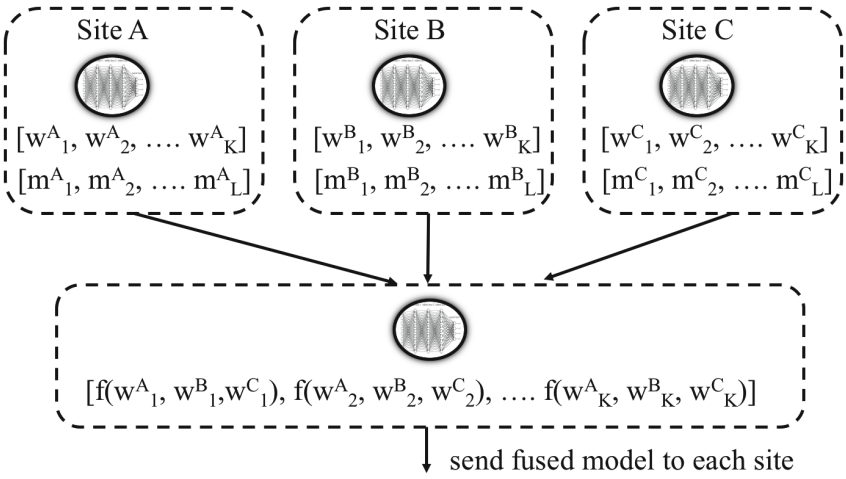


Fig. 7. Federated learning synchronization step

operation running several orders of magnitude slower than the unencrypted operation, which makes its use in a real-world situation difficult. Instead of FHE, we will use partial homomorphic encryption or PHE to only send encrypted parameters for fusion to the model fusion server.

Since any PHE operation can only perform one of the mathematical operations, multiplication or addition, on encrypted information and still preserve the homomorphy, we will need to restructure the operations at the agents and the federation server so that they only use one type of operation. Each of the local sites performs the other operation needed for the fusion process, with the fusion server helping with the coordination aspects. We can use any PHE mechanism (such as Paillier [12] and El Gamal [13]) for encryption, each of which are homomorphic with respect to addition.

In order for PHE operations to work properly, each of the agents ought to be using the same parameters. Thus, a key generation and parameter exchange process needs to be in place for the agents to use consistent keys. The shared keys cannot be created by the federation server, since it can possibly use the knowledge of the key to decrypt the model parameters. One of the agents will have to take over the task of generating the shared keys and ensuring that it reaches the other agents in the system. We call this agent the leader agent.

In order to select the leader, a distributed leader election protocol [14] can be followed. However, distributed leader election protocols are complex, and it will be easier to use a client-server centric approach for electing the leader. This simplified leader election can be enabled by the federation server.

The resulting mechanism follows a three stage process as shown in Fig. 8. In the first stage, the federation server acts as the central location where the federation agents can register and indicate their presence along with their attributes. A set of policies are used by the federation server to determine one of the agents as

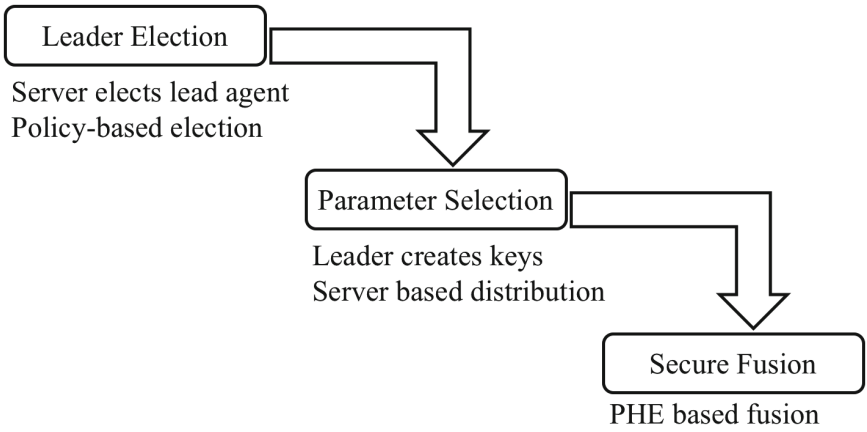


Fig. 8. Process for secure model fusion

the leader. While the federation server and the federation agents are in different trust zones, each of them can check whether the federation server had done the selection in accordance with the specified policies. Using a centralized federation server for this election would be more efficient than a distributed leader election process among the different agents.

In the second stage, the elected leader selects the common keys that can be used for partial homomorphic encryption. These keys are shared securely among all the agents using public key cryptography. The server can assist in the dissemination of the keys but never gets to see the keys itself. In the third stage, the different agents work together with the federation server using the commonly selected keys for partial homomorphic encryption.

Looking at the process in more detail, the agents all register with the fusion server in the first stage. Each agent generates a public/private pair of keys, with agent i creating a pair (Pk_i, Sk_i) for secure communication. The agent then provides its public key/certificate to the fusion server. The fusion server can send the public keys to any agents that join the fusion session. These public keys are used to establish secure communication between the agents and the fusion server, and for agents to communicate with each other securely.

The fusion server keeps track of the fusion session (i.e., counts the number of agents in the fusion process, keeps track of their public keys) and it also selects one of the agents as the leader for the fusion process. Based on the set of policies provided to the federation server, the elected leader could be the first agent that joined a federation session, or it could be the agent with the best possible computational resources, or the agent with the maximum amount of data.

In the second stage, the leader agent selects the hyper-parameters for the federated model learning. It uses the public key of other agents to send this information securely to each of the agents via the fusion server. It also selects

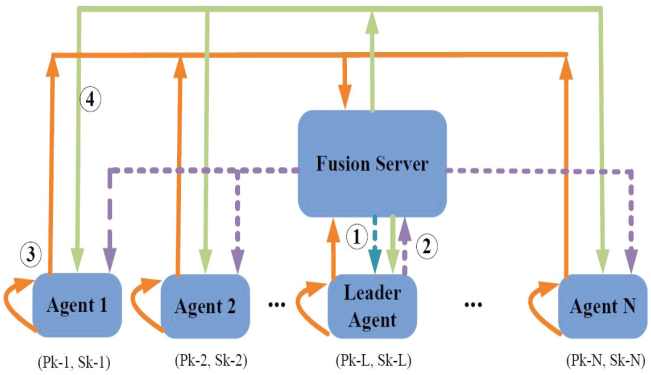


Fig. 9. Information exchange in secure model fusion

the type of the PHE algorithm to be used, and the parameters required for it. This information is distributed securely to other agents via the fusion server.

During each step of the training process, the agents use the same homomorphic encryption algorithm and the same parameters to encrypt the weights before sending them over to the fusion server. If required, each agent would locally compute the product of its priority p_i and each of the entries in the model weight vector $\boldsymbol{w}_i = [w_{i1}, w_{i2}, \cdots, w_{ik}]$. Each of these entries are then encrypted and sent to the fusion server.

The fusion server performs an element-wise multiplication on the weights of the encrypted model that it gets from each of the agents, and then sends the vector of this multiplication back to the agents. Each agent decrypts the results and then recreates the model parameters of the fused model system.

By using this strategy, the fusion server never sees the raw model parameters, and can utilize the additive homomorphic property of PHE to fuse the models together.

We show the whole process for secure model fusion in Fig. 9. In step 1, the public/private key registration is computed between each agent and the fusion server. The fusion server selects a leader agent for coordinating parameter setup between agents and fusion server. In step 2, the leader agent selects the hyper-parameters of the learning including the types and parameters of the PHE method and the priority for each agent, which are then distributed securely (encrypted with each agent's public key) to each participating agent by the fusion server. In step 3, each agent computes its local model and sends the encrypted local model (with priorities) to the fusion server. In step 4, the fusion server performs multiplication over the encrypted models and sends the encrypted model back to each participating agent. The process from Step 3 to Step 4 would repeat until convergence.

4 Performance Considerations

While the accuracy of federated learning is comparable to that of centralized learning, the use of partial homomorphic encryption raises concerns about the performance of the system. One specific concern is the increase in the training time that the system would incur. This increase in the training time depends on the relative ratio of the time it takes to encrypt the weights of the model, versus the time it takes for the unencrypted training.

Figure 10 illustrates the time-line by which federated learning happens. Each of the agents trains a pre-negotiated neural network over a mini-batch of the data that it has. After each mini-batch, the different sites exchange information with each other for synchronization and to average the weights that they have computed. After this synchronization, they move on to the next mini-batch of training data.

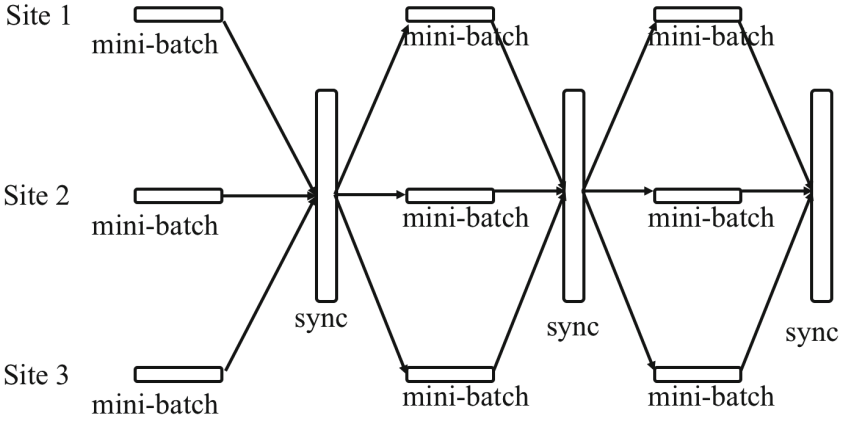


Fig. 10. Steps in federated learning

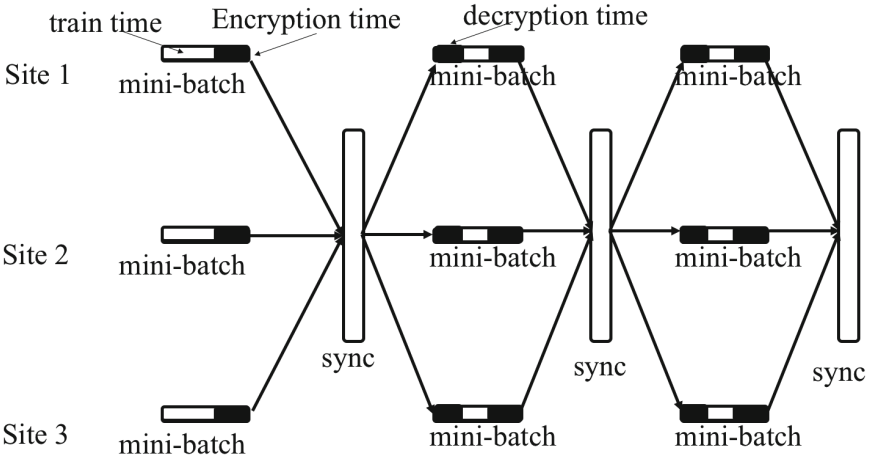


Fig. 11. Steps in secure federated learning

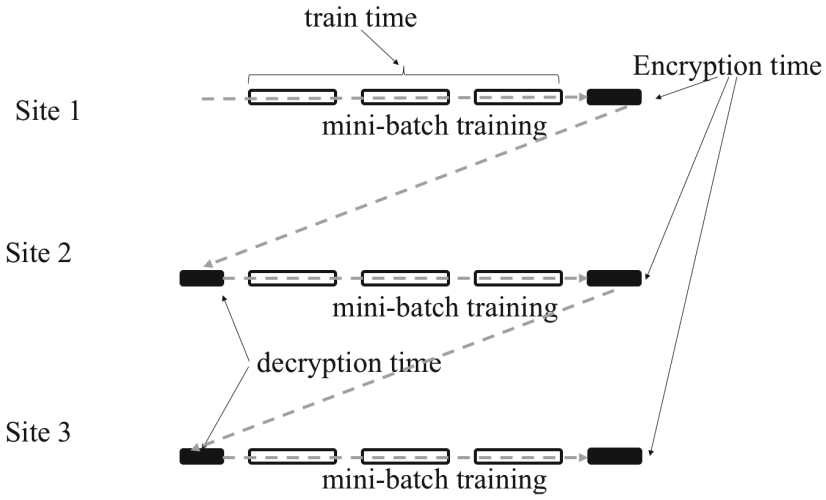


Fig. 12. Steps in secure federated learning using model movement approach

When PHE is employed, an encryption of the weights is done prior to the synchronization. This adds an additional overhead. The process is illustrated in Fig. 11, and the encryption time is shown as a dark shaded area after the white training period. After the synchronization happens, each site receives the fused model and has to decrypt the weights before proceeding to training on the next mini-batch. This results in an increase in the overall time taken for the model to execute.

The degradation in performance depends on the ratio of the time it takes to train on a mini-batch compared to the time it takes to encrypt the parameters and decrypt them on receipt. This ratio depends on several factors, the size of the mini-batch, the number of parameters in the model, and the network synchronization latency. The training time increases according to the ratio, with the network synchronization time remaining unchanged. A larger network synchronization time mitigates against the increase caused by the encryption.

There are several flavors of federated learning, and the overhead of partial homomorphic encryption would be different depending on the flavor used. In addition to the synchronized model for federated learning, another flavor can be used in which the training is done over all the mini-batches [3], and the model moved over to another site once the training is done. The encryption overhead in this case is negligible compared to the time taken to train the model at each site, and it results in very little overhead. This process with encryption added in is shown in Fig. 12. The dashed gray line in the figure shows how the training happens over the mini-batches distributed over different sites, interspersed with the encryption of the parameters and the decryption at the next site. The encryption step happens after the training finishes at each site, while the decryption step happens at the beginning of the training of each site.

With model migration, the homomorphic encryption needs to be performed only once after all the data at any site has been used for training the model, and the parameters are decrypted at the next site after it is moved there. In another variation, the models are trained in parallel at each site, and the weights are averaged together after the models are received at the federation server. When trained for multiple such rounds, this approach also results in a performance that is comparable to that of the centralized model.

We can create a simple model to compare the performance of secure federated learning to the standard model based on the steps shown in Fig. 11. If we define T_l as the time spent in the machine learning process at a site between encryptions, E as the time to encrypt a single number using the PHE algorithm, and to decrypt the same number back, W as the number of weights in the neural network being trained on, and T_n as the time involved in the network synchronization, the amount of time a secure model takes to complete compared to federated learning in the clear would equal

$$(T_l + E \cdot W + T_n)/(T_l + T_n),$$

which is equal to

$$1 + E \cdot W/(T_l + T_n).$$

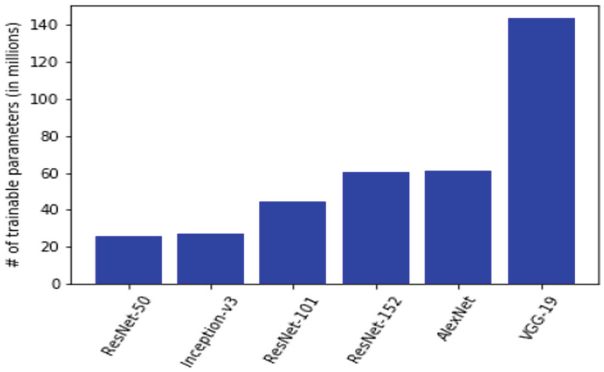


Fig. 13. The number of parameters for 6 models (ResNet-50, Inception-v3, ResNet-101, ResNet-151, AlexNet, VGG-19).

The amount of overhead incurred depends on the number of parameters that are included in the neural network model. For models built on relational data, which usually consists of less than 100 weights, the overhead may be relatively minor. An increase in the size of the mini-batches will result in improving the overall performance.

On the other hand, in the domain of image classification, where convolutional neural networks can easily have thousands of weights, the overhead may be substantially higher. Figure 13 shows the number of parameters for 6 popular

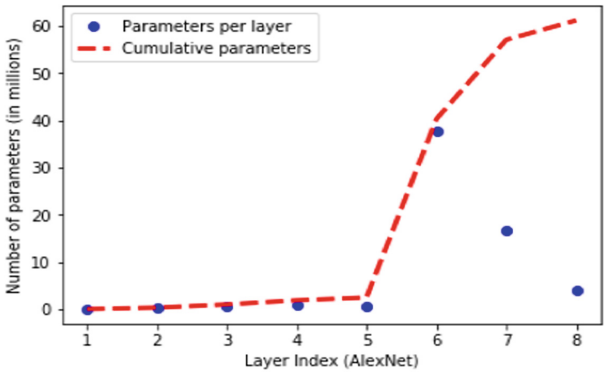


Fig. 14. The number of trainable parameters per layer of the AlexNet model.

models for image processing (ResNet-50, Inception-v3, ResNet-101, ResNet-151, AlexNet, and VGG-19). Figure 14 shows the number of trainable parameters per layer of the AlexNet model. From both figures, we know that the models for image classification are usually associated with large numbers of weights, thus making the element-wise encryption/decryption time-consuming. If we need to use secure federated learning with images, we need to develop techniques that can reduce the overhead of PHE encryption.

In the next couple of sections, we discuss some of the techniques that can reduce the overhead to make the scheme practical for networks with large numbers of weights.

5 Optimization for Practical Realization of Secure Distributed Systems

Depending on the number of weights in the neural network model (e.g., a multilayer network with dense connections will have more weights), the computation time taken for encrypting and decrypting the weights can dominate the training time of the secure federated learning system. To reduce the overall training time for large models, we propose two optimization strategies representing different trade-off regimes between model accuracy and encryption/decryption speed. The first strategy is *quantization of model weights*. By limiting the precision of model weights, we can construct two lookup tables (for encryption and decryption respectively) of the pre-computed encryption/decryption pairs for each possible model weight. In the training process, we encrypt each locally computed model weight by looking up its encrypted value in the encryption table. The fused model weights securely computed by the server can be similarly decrypted by each agent by use of the decryption table. The second strategy is *batch encryption/decryption of model weights*, i.e., implementing encryption/decryption for a series of model weights by batching them together. Next, we discuss these two optimization strategies in detail and evaluate their effectiveness using experiments.

5.1 Quantization of Model Weights to Construct Lookup Tables

Model weights are typically represented using floating-point numbers (which allows for greater precision and broader range). However, training algorithms used for deep neural networks have been shown to be resilient to the error introduced by using low precision fixed-point representations [15]. Advantages of using fixed point representations are faster hardware operations and reduced memory footprint. Disadvantages include loss in model accuracy, and a reduced range of the represented numbers. Motivated by [15], we propose to leverage the natural resilience of deep neural networks to low-precision fixed-point representations for speeding up the encryption and decryption process. Quantization can be performed by rounding the model weights. For a given level of precision, one can pre-compute the encryption/decryption pairs for every number in the range and construct the encryption/decryption lookup tables accordingly. During model training, constant time encryption and decryption can be performed using the look-up tables.

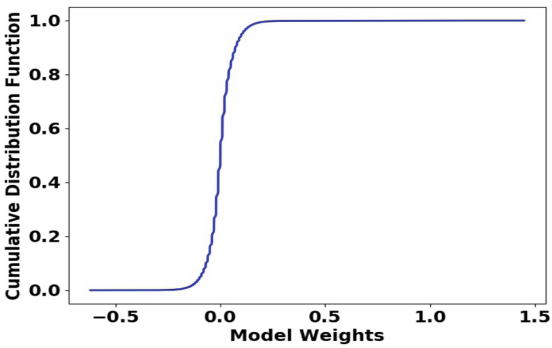


Fig. 15. CDF of model parameters in the training process.

Experimental Setup: We conduct experiments to validate the effectiveness of this optimization mechanism. In our experiments, we consider a scenario where 2 agents aim to train a neural network for digit recognition using the MNIST dataset [16], which has 60,000 training data elements and 10,000 testing data elements. The training dataset is randomly distributed among the two agents and the size of each training sample is 28×28. We set the initial learning rate as 0.003 and the learning rate decay as 0.9, and aim to minimize the cross-entropy loss. Furthermore, we consider two models representing a complicated model and a simple model, respectively. The first model has one hidden layer with 784 neurons (Model I), i.e., fully connected network, and the second model also has one hidden layer with 20 neurons (Model II). In the optimization process, we quantize each model weight to a fixed level of precision (with a limited number of decimal places) and construct the lookup tables for the encryption/decryption pairs using the El Gamal algorithm [13]. Our experiments were conducted in the

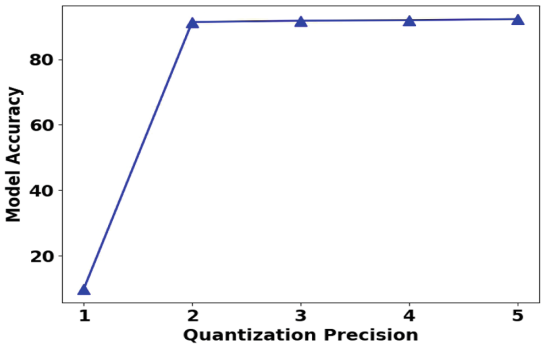


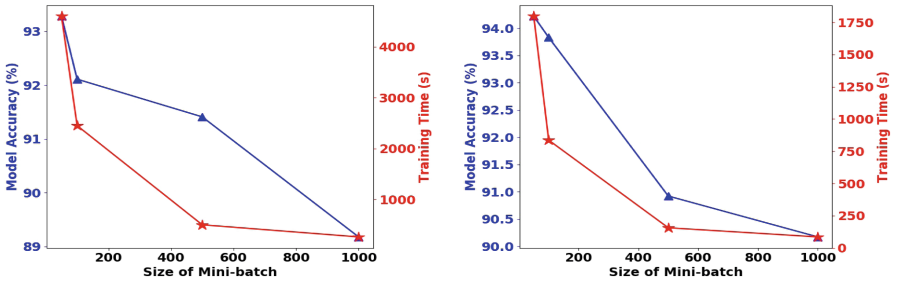
Fig. 16. Model accuracy under different precision of model weights (x-axis represents the number of decimal places in the model weights).

Python programming language on a PC with a 2.9 GHz Intel Core i7 processor and 16 GB of memory.

Intuition of Constructing Lookup Tables: We first show the cumulative distribution function (CDF) of the model weights in the distributed learning process in Fig. 15, where we set the size of each mini-batch as 50 and the number of epochs as 2 under Model I. From Fig. 15, we observe that most weights are concentrated within $[-0.25, 0.25]$, which provides the foundation for our optimization scheme through weight quantization.

Influence of Model Weight Quantization: To construct the lookup table, we aim to limit the precision of the model weights by rounding them to a fixed number of decimal places. In Fig. 16, we show the model accuracy under different precision levels (by rounding model weights to $1, 2, 3, 4, 5$ decimal places). From Fig. 16, we observe that rounding model weights to 2 decimal places is enough to maintain high model accuracy. Therefore, in our following experiments, we construct the encryption/decryption lookup table by pre-computing the encrypted values for all numbers $\in [-1, 1]$ with 2 decimal places.

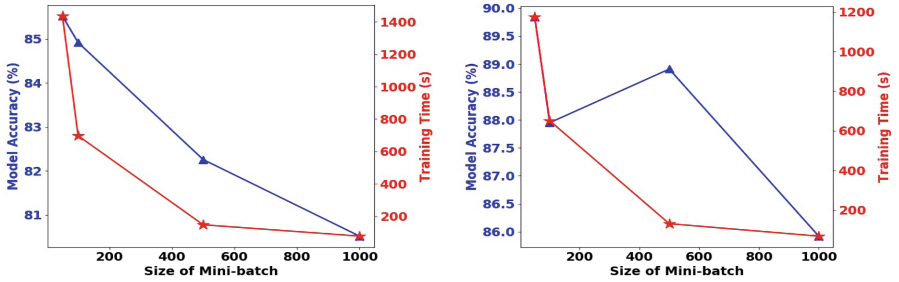
Trade-off Between Model Accuracy and Training Time with Varying Sizes of Mini-batch: In Fig. 17, we show the performance of our secure distributed learning and the unencrypted distributed learning with varying sizes of mini-batches under Model I. Specifically, we set the number of epochs as 1 and quantize each model weight to 2 decimal places. Similarly, we show the trade-off between model accuracy and training time under Model II in Fig. 18. From Figs. 17 and 18, we have the following important observations: (1) a smaller size of mini-batch would result in a higher accuracy while at the cost of a longer training time; (2) comparing Fig. 17(a) and (b) (also Fig. 18(a) and (b)), we know that the quantization of model weights would degrade the model accuracy



(a) Accuracy-Time Trade-off of Secure Distributed Learning

(b) Accuracy-Time Trade-off of Unencrypted Distributed Learning

Fig. 17. The trade-off between accuracy and training time of secure distributed learning and unencrypted distributed learning with varying sizes of mini-batch under Model I.



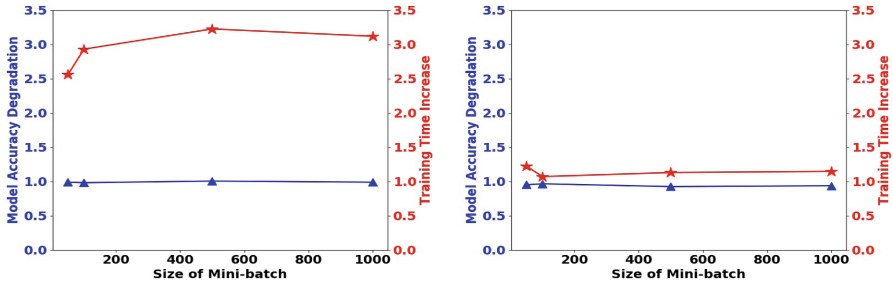
(a) Accuracy-Time Trade-off of Secure Distributed Learning

(b) Accuracy-Time Trade-off of Unencrypted Distributed Learning

Fig. 18. The trade-off between accuracy and training time of secure distributed learning and unencrypted distributed learning with varying sizes of mini-batch under Model II.

and the encryption/decryption implementation would increase the training time; (3) comparing Figs. 17 and 18, we know that different models have various influence on balancing model accuracy and training time. For instance, the accuracy degradation under Model II (≈5%) is much more serious than that of Model I (≈1%), indicating that the influence of model weight quantization has a more significant effect on this simpler model.

In Fig. 19(a) and (b), we show the accuracy degradation ratio and training time increase ratio of our secure distributed learning system compared to the unencrypted learning system for Model I and Model II, respectively. From Fig. 19(a), we observe that the encryption/decryption implementation incurs up to $3x$ overhead in training time while there is negligible accuracy degradation under Model I, validating the effectiveness of this optimization strategy. From Fig. 19(b), we observe that the additional training time under Model II is much lower than that of Model I.



(a) Accuracy Degradation Ratio/Time Increase Ratio under Model I

(b) Accuracy Degradation Ratio/Time Increase Ratio under Model II

Fig. 19. Accuracy degradation ratio and training time increase ratio with varying sizes of mini-batch under Model I and Model II.

For a secure federated learning system, where the agent models are fused after every mini-batch, increasing the size of the mini-batch can reduce the number of encryption and decryption cycles improving training time (shown in Figs. 17(a) and 18(a)). However, a very large mini-batch size have been shown to adversely effect the training accuracy of the model. Therefore, depending on the training data and the model parameters, an appropriate mini-batch size needs to be selected. One can employ a separate search algorithm to obtain the right mini-batch size such as grid search or Bayesian optimization [17].

Trade-off Between Model Accuracy and Training Time with Varying Number of Epochs: The number of training epochs also has a direct effect on the training time and accuracy of the model. Figure 20 shows the trade-off between model accuracy and training time of our secure distributed learning system with different number of training epochs under Model I (the size of mini-batch is set to 500). From Fig. 20, We observe that the accuracy improves a bit with more epochs, while the computational time is almost linearly increasing with the number of epochs. A proper number of training epochs needs to be selected to obtain a desired trade-off between model accuracy and training time.

Limitation: Although this optimization mechanism has shown effectiveness in experiments (Fig. 19), there still exist scenarios where the construction of lookup tables may not be feasible. There is *randomness* in the PHE mechanisms and we exploit the same randomness in each training process (which could vary in the next training process). In this manner, the security of such encryption/decryption processes may not be that strong as scenarios where the randomness varies for each message. For instance, the same randomness in the training process (which allows for constructing the lookup table) may make the training model/data suffer from the *chosen ciphertext attack*. How to construct the lookup table while enhancing the security of the PHE mechanism (e.g., with arbitrary randomness per message) is an interesting problem for future research.

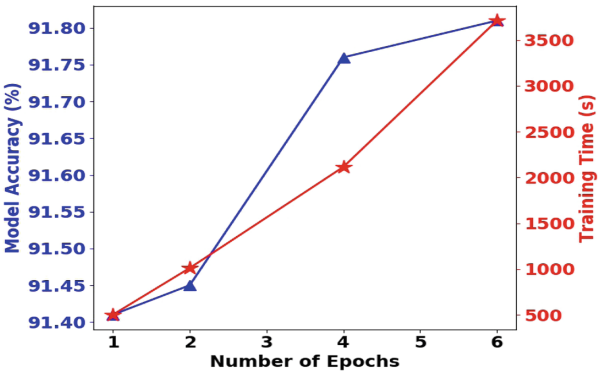


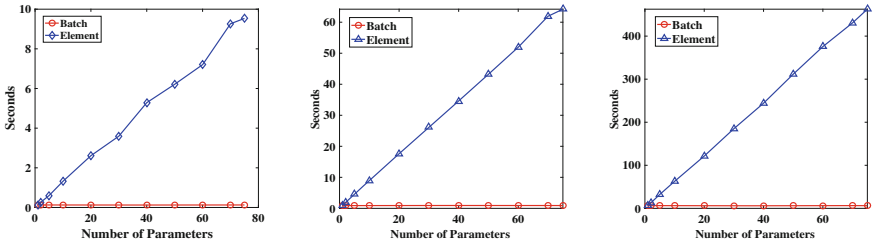
Fig. 20. The trade-off between accuracy and training time of secure distributed learning and unencrypted distributed learning with varying sizes of mini-batch under Model I.

5.2 Batch Encryption of Model Weights

Most existing PHE methods [12,13] encrypt only a single scalar at a time. Therefore, an intuitive approach for secure model fusion by applying PHE to each element in the local model vector is computationally expensive. We propose another optimization strategy by leveraging batch encryption that can encrypt/decrypt a group of weights in one operation. Our batch encryption algorithm combines quantization and zero-padding techniques to enhance computational efficiency without degrading the learning accuracy. To adopt PHE to work with batches of model weights at one time, we need to concatenate multiple weights together prior to the encryption process and then separate them apart after the decryption process.

For a locally trained model $p_i \boldsymbol{w}_i$, we first quantize the precision for each element to only keep r numbers of decimal places. Next, we pad zeros prior to the quantized element to keep its overall length as l. Then, we concatenate all the weights into one value and then apply PHE for one-time encryption of the weighted local model, i.e., $Enc_{batch}(p_i \boldsymbol{w}_i)$. The fusion server then applies multiplication over all the local models as $\prod_i Enc_{batch}(p_i \boldsymbol{w}_i)$. Next, the encrypted fused model is transmitted to each agent for decryption as $Dec(\prod_i Enc_{batch}(p_i \boldsymbol{w}_i))$, which would be divided into multiple l-digit scalars corresponding to each fused model weight. Based on the decrypted model weights, each agent will update its own local model and repeat the whole process until convergence.

Advantage of Batch Encryption: In this section, we aim to first show the advantage of our proposed batch encryption mechanism over the baseline element-wise approach. Specifically, we consider the Paillier method [12] and evaluate the computational time with respect to the size of model weights while fixing the size of key pairs as $KeySize = 2048$ in Fig. 21(a). We can observe that the computational time of the baseline approach increases linearly with the num-



(a) KeySize = 2048 (b) KeySize = 4096 (c) KeySize = 8192

Fig. 21. Computational complexity of our proposed batch encryption and the baseline element-wise encryption

ber of the model weights. In comparison, our proposed batch encryption method ($r = 5,\ l = 8$) causes negligible additional computation with a larger number of model weights, which shows the superiority of this optimization strategy. For safely storing the integer and detecting integer overflow, we only consider to concatenate every 75 weights into one value as shown in Fig. 21(a). Therefore, we can conclude that this batch encryption mechanism can achieve up to $75x$ improvement in computational complexity over the baseline approach.

To evaluate the performance under different lengths of the key in Paillier, we show the results under $KeySize = 4096$ and $KeySize = 8192$ in Fig. 21(b) and (c), respectively. We observe that a longer key would incur more computations, which at the same time is more secure to defend against key inference attacks. In practice, it is important to select a proper key size for achieving balance between model security and training time.

Training Time Estimation for Batch Encryption: Next, we analyze the overall complexity for using this batch-encryption optimization mechanism. We assume that there are N_{train} data in the model training process, N_{agent} agents each with N_{train}/N_{agent} training data, the number of model weights is N_{model}, the size of mini-batch is S_{batch} and the number of epochs is N_{epoch}. Therefore, the gradient fusion process for each epoch is repeated for $N_{train}/N_{agent}/S_{batch}$ times. Then, we have the overall operations of implementing PHE encryption and decryption process as $N_{epoch} \times N_{model} \times N_{train}/N_{agent}/S_{batch}$ and $N_{epoch} \times N_{model} \times N_{train}/N_{agent}/S_{batch}/75$ for the baseline approach and the batch encryption approach, respectively. Denote the computational time for one-time encryption and decryption process in PHE frameworks under a given $KeySize$ as $T_{KeySize}$. Therefore, the additional computational time incurred by the baseline approach and our batch encryption approach can be computed as

$$\begin{aligned} T_{baseline} &= N_{epoch} \times N_{model} \times N_{train}/N_{agent}/S_{batch} \times T_{KeySize} \\ T_{batch} &= N_{epoch} \times N_{model} \times N_{train}/N_{agent}/S_{batch}/75 \times T_{KeySize} \end{aligned} \tag{1}$$

Since the $N_{epoch}, N_{model}, N_{train}, N_{agent}, S_{batch}$ would significantly affect the accuracy of the training model and $T_{KeySize}$ determines the security performance

of the model, we know that a trade-off between model accuracy, model security and computational time can be achieved by properly selecting these system parameters.

Limitation: Although the training time for this batch optimization mechanism can decrease that of the baseline approach by $75x$, the overall training time might still be high especially for models with large numbers of weights. Therefore, this batch optimization mechanism is more suitable for shallow models built on relational data. For high-dimensional input data and complicated models, the first optimization mechanism using encryption/decryption lookup tables (Sect. 5.1) is likely to achieve better performance.

5.3 Discussion on Practical Implementation

- The first optimization strategy of leveraging encryption/decryption lookup tables can greatly reduce the time consumed in encrypting/decrypting each model parameter. For instance, our experiments demonstrate that the training time of secure distributed learning is only increased by $3x$ compared to that of unencrypted distributed learning, while the accuracy of the training model is not seriously degraded.
- The second optimization strategy of leveraging batch encryption can greatly reduce the training time without affecting model accuracy. Our experiments demonstrates that the batch-encryption optimization mechanism can achieve up to $75x$ improvement in training time over the baseline approach.
- In practice, we need to select a proper optimization method by comprehensively analyzing the training process and carefully selecting appropriate values of system parameters including the size of mini-batch, the number of training epochs, etc. These provide interesting opportunities for practical design of secure distributed learning systems.

6 Related Work and Threat Model

In this section, we first briefly discuss the frameworks of distributed learning and partial homomorphic encryption, as well as related works regarding these two topics. Then, we describe the threat model considered in our work.

6.1 Distribution Learning

The proliferation of big data has boosted the development of distributed machine learning [1,2,4–8]. Distributed machine learning aggregate the computational capability of multiple agents to achieve accurate model training performance. Specifically, each agent trains a learning model locally which is then transmitted to the central server for aggregation, and this process is repeated at each training step. Within distributed machine learning, Federated learning [1,2] aims to enable a collaborative learning among different agents using their local training data without the need to store user data in the cloud. Therefore, the models can be learnt from each other without exposing private data sources.

6.2 Partial Homomorphic Encryption

Homomorphic encryption scheme enables computation (addition and multiplication) in the encrypted domain. Fully homomorphic encryption (FHE) [11] is capable of encrypting a value that can be added and multiplied by another encrypted value. Partially homomorphic encryption (PHE) [12,13], on the other hand, can only perform either addition or multiplication. The Paillier cryptosystem [12] is one of the most popular PHE mechanism which utilizes public key encryption scheme while preserving the *additive* property. Therefore, the product of two ciphertexts will decrypt to the sum of their corresponding plaintexts, i.e., $Dec(Enc(m_1) \times Enc(m_2)) = m_1 + m_2$ for any two plaintexts m_1, m_2. Both FHE and PHE can be applied in the distributed learning setting to protect privacy of the computed model and the locally stored data. However, FHE mechanisms are usually computationally complicated making it difficult to be applicable for reality. Therefore, we consider to apply PHE for secure model fusion in distributed learning in order to achieve a better balance between security and usability.

6.3 Privacy-Preserving Distributed Machine Learning

Threat Model: In our setting, we consider all the other agents and the fusion server as potential adversaries that are honest-but-curious. Our mechanism aims to protect both the computed models and the locally stored data which are sensitive information of the users.

Previous work in [18] encrypted the locally computed models to protect the individual data from the server while the server can still access the aggregated model, based on their assumption that the aggregate model does not disclose any sensitive information about individual data. However, recently proposed model inversion attacks [19] and membership inference attacks [9], have shown that individual data can be inferred from the aggregate model. That is to say, the direct exposure of the aggregate model to the server would put the privacy of individual data into risk. Therefore, the protection for the computed model should also be taken into consideration in the design of secure model fusion system as in our setting.

Secure multi-party computation (SMC) [20] aims to obtain correct output computed over multiple agents even if some agents may misbehave. In SMC, if one agent learns the output then all honest parties can learn the output. Therefore, it is different from our setting where we only consider honest-but-curious agents in distributed learning and the fused model can not be exposed directly to these agents for security reasons.

7 Conclusion

In this paper, we have proposed a secure model fusion mechanism in federated learning systems, in order to protect the computed models and the corresponding training data. By leveraging the *additive* property of PHE mechanisms, we

convert fusion operation into portions where the fusion server can update the model over the encrypted local models. We further propose two optimization mechanism that can decrease the computational overhead and make the scheme practical even for networks with large number of model weights.

Although we have presented our work using El Gamal [13] and Paillier [12], it can be generally applied to other PHE/HE methods in a straightforward manner. Our future work includes balancing model accuracy, model security and computational time by properly selecting system parameters.

References

1. McMahan, H.B., Moore, E., Ramage, D., Hampson, S.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
2. Bonawitz, K., et al.: Practical secure aggregation for federated learning on user-held data. arXiv preprint arXiv:1611.04482 (2016)
3. Verma, D., Julier, S., Cirincione, G.: Federated AI for building AI solutions across multiple agencies. In: AAAI FSS-18: Artificial Intelligence in Government and Public Sector, Arlington, VA, USA (2018)
4. Wang, S., et al.: When edge meets learning: adaptive control for resource-constrained distributed machine learning. In: IEEE International Conference on Computer Communications (2018)
5. Verma, D., Chakraborty, S., Calo, S., Julier, S., Pasteris, S.: An algorithm for model fusion for distributed learning. In: Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, vol. 10635, p. 106350O. International Society for Optics and Photonics (2018)
6. Li, M., et al.: Scaling distributed machine learning with the parameter server. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI), vol. 14, pp. 583–598 (2014)
7. Kraska, T., Talwalkar, A., Duchi, J.: MLbase: a distributed machine-learning system. In: 6th Biennial Conference on Innovative Data Systems Research (CIDR 2013) (2013)
8. Dean, J., et al.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems, pp. 1223–1231 (2012)
9. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE (2017)
10. Long, Y., et al.: Understanding membership inferences on well-generalized learning models. arXiv preprint arXiv:1802.04889 (2018)
11. Gentry, C.: A fully homomorphic encryption scheme. Stanford University (2009)
12. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
13. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985)
14. Nakano, K., Olariu, S.: A survey on leader election protocols for radio networks. In: Proceedings. International Symposium on Parallel Architectures, Algorithms and Networks, I-SPAN 2002, pp. 71–76. IEEE (2002)

15. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning, pp. 1737–1746 (2015)
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
17. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
18. https://blog.n1analytics.com/distributed-machine-learning-and-partially-homomorphic-encryption-1/
19. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322–1333. ACM (2015)
20. Goldreich, O.: Secure multi-party computation. Manuscript. Preliminary version **78** (1998)