2023 20th Annual International Conference on Privacy, Security and Trust (PST) | 979-8-3503-1387-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/PST58708.2023.10320176

A Secure Distributed Learning Framework Using Homomorphic Encryption

Stephen Ly
Department of Computer Science
California State University, Sacramento
Sacramento, CA, USA
stephenly.tmp@gmail.com

Yuan Cheng
School of Computing
Grand Valley State University
Allendale, MI, USA
yuan@ycheng.org

Haiquan Chen
Department of Computer Science
California State University, Sacramento
Sacramento, CA, USA
haiquan.chen@csus.edu

Ted Krovetz
Department of Computer Science
California State University, Sacramento
Sacramento, CA, USA
tdk@csus.edu

***Abstract*—The increasing complexity of artificial intelligence (AI) models poses a significant challenge for individuals and organizations without sufficient computing resources to train them. While cloud-based training services can offer a solution, they require sharing sensitive data with untrusted parties, posing risks to data privacy. To address this challenge, we explore the combination of distributed training and homomorphic encryption to parallelize the training process on encrypted data. We utilize the CKKS homomorphic encryption scheme to develop a framework that can train comparably accurate AI models in less time than other homomorphically encrypted training solutions. Our experiments demonstrate reduced total runtime for homomorphically encrypted model training while maintaining competitive classification accuracy for the MNIST handwritten digits dataset, a well-known benchmarking dataset for machine learning. Our framework brings homomorphic encryption closer to becoming a practical data privacy solution for small stakeholders who cannot afford to compromise on security.**

***Index Terms*—distributed learning, homomorphic encryption, privacy-preserving machine learning**

I. Introduction

Artificial intelligence (AI) and its models play a critical role in various technologies today, including computer vision, search recommendations, and autonomous vehicles. However, the growing complexity of these models necessitates more computational power to train them. As a result, small businesses and individuals without access to dedicated data centers cannot match the scale of tech companies, leading to a higher cost of entry. Cloud computing services, such as Amazon Web Services (AWS) and Google Cloud Platform (GCP), have emerged to fill this gap by providing outsourced computational power. These platforms allow for the use of distributed learning, a machine learning approach that parallelizes training over a large number of machines. This method offers several advantages over traditional single-machine training, including scalability, efficiency, and cost-effectiveness [1].

While distributed learning has become popular, it presents unique challenges. For example, model training outside one's own machine and network can expose sensitive data to malicious actors and lead to security breaches. Factors like medical privacy laws and personally identifiable information may also render a dataset confidential. These data security and privacy concerns have resulted in over a third of organizations stopping or slowing down cloud service adoption [2]. Therefore, securing sensitive information before using such datasets for model training in a distributed learning system is crucial.

Several solutions have been proposed to address data privacy concerns in distributed learning, such as federated learning [3], differential privacy [4], and cryptographic protocols like secure multiparty computation [5] and homomorphic encryption [6]. Federated learning and secure multiparty computation are effective for collaborative training, where sensitive data remains on the host machine throughout the process. However, they do not alleviate the computational burden on individuals wanting to train their models. In contrast, differential privacy and other obfuscation techniques allow the data to leave the host for outsourced training. However, the level of security provided by these techniques (e.g., adding noise to the dataset) is inversely proportional to their usability.

Homomorphic encryption is a promising solution for ensuring data privacy in distributed learning. Unlike traditional encryption schemes, homomorphic encryption allows basic arithmetic operations to be performed directly on encrypted data, eliminating the need for decryption and re-encryption. However, homomorphic encryption is known for its heavy computational overhead, hindering its applicability in real-world scenarios [7]. This overhead can increase training runtime by several orders of magnitude on a single machine.

To address these challenges, we propose a secure distributed learning framework that combines the advantages of distributed learning and homomorphic encryption while mitigating the drawbacks of each. Our framework enables better performance over single-machine training using homomorphic encryption and alleviates data privacy concerns related to distributed learning. Specifically, we use CKKS

[8], a homomorphic encryption scheme that operates on real numbers, to secure sensitive datasets before model training. This allows us to encrypt datasets without converting data into fixed-point integers. The main contributions of our project are as follows:

- We design and build a distributed learning framework that uses homomorphic encryption to secure sensitive data, allowing users to train AI models securely with no perceivable loss of model accuracy.
- We demonstrate the potential speedup achieved by utilizing distributed learning techniques to distribute the workload of homomorphic encryption.
- We discuss how we design the loss evaluation method to mitigate a chosen ciphertext attack on result vectors.

Our experimental evaluation shows competitive accuracy results on the MNIST handwritten digits classification problem [9], achieving a classification accuracy comparable to an identical model trained using a similar encryption approach [10]. Moreover, our runtime tests indicate that our framework significantly improves as the number of machines increases, reducing the training time from several days to hours. These experiments also reveal that network overhead takes up a small fraction of the total runtime, suggesting the potential for further investigation and advancements in networked solutions.

The remainder of this paper is organized as follows. Section II reviews prior literature on data privacy in distributed learning and various existing approaches to address the problem. In Section III, we provide an overview of the methodology and framework of our solution, including its design philosophies and key components. Section IV provides detailed specifications of the framework, including information on the actors, algorithm specifics, and dataflow. Section V outlines the experimental evaluation of our framework, including the test case, inputs, and parameters. Section VI presents the results of our experiments and compares our work to related research. In Section VII, we discuss the challenges encountered and the insights gained from the project. Finally, in Section VIII, we conclude the paper with our final thoughts and highlight areas for future work.

II. Related Work

Data privacy in distributed learning has received significant attention in recent years. The two main categories of techniques used to secure data in distributed learning systems are system-level architectures with built-in privacy assurances and data obfuscation algorithms. Examples of the first category include federated learning platforms and secure multiparty computation frameworks, while the second category includes differential privacy and homomorphic encryption.

Federated learning systems, as introduced by McMahan et al. [3], enable the training of a global model using local data from each party without transferring the raw data to a central server. This decentralized approach to data training grants strong data privacy assurances for data owners participating in the algorithm, making it useful for machine learning on sensitive data [11]. Several papers have expanded on the security of federated learning by incorporating data obfuscation techniques, such as differential privacy and homomorphic encryption. For example, Geyer et al. combined federated learning with differential privacy, creating a framework that adds noise to client data based on the sensitivity of the data [12]. BatchCrypt by Zhang et al. used homomorphic encryption to encrypt gradients instead of raw data, thereby preventing information leaks due to gradient analysis on the server side [13]. However, these systems are designed for training a centralized model on decentralized data, which is different from our project's use case, where both data and model are centralized.

Differential privacy is another technique used to protect sensitive information by obfuscating the actual dataset. It achieves this by adding random noise to queries, preventing the identification of individual data points. Abadi et al. demonstrated the usage of differential privacy in deep learning, showing how it can be applied to protect gradients [4]. Wang et al. applied noise to input features based on their importance, thus minimizing the impact on model utility when using differential privacy for model training [14]. Because differential privacy is much less computationally intensive than homomorphic encryption, it is also a popular choice for federated learning, as seen in Geyer et al. and Hu et al.'s federated learning frameworks [12], [15]. However, there is a distinct tradeoff between the level of privacy and model accuracy in machine learning when using differential privacy. For datasets that include images and other perceivable patterns, considerable noise may be required to obfuscate information properly, at which point the accuracy of the AI model suffers as well [16].

Several deep learning systems using homomorphic encryption have been proposed [17], [18], [19]. Al Badawi et al. presented a GPU-accelerated homomorphic convolutional neural network (HCNN) that employed the BFV scheme [17]. Although this HCNN reported a 5.16-second inference time with 99% accuracy on the MNIST dataset, the authors noted a security level greater than 80 bits, which falls on the lower end for secure encryption algorithms. Ishiyama et al.'s convolutional neural network using homomorphic encryption introduced the use of ReLU (rectified linear units) [20], [21] and Swish [22] approximations for activation functions, allowing for the direct use of the CKKS scheme in model inference [18]. Sphinx, a deep learning system developed by Tian et al., adopted a client/server architecture designed to safeguard the privacy of AI training in networked environments [23]. Recognizing the potential threats to data privacy from network snooping or intrusive servers, Sphinx combined homomorphic encryption with differential privacy to protect data during the training and inference phases. However, Sphinx's implementation did not leverage a distributed learning network, which restricts the scalability of their solution. Prior literature has mainly focused on the inference stage of AI model deployment. Research into homomorphic encryption in model training is sparse, with Nandakumar et al.'s work on neural network training on encrypted data being a notable example [10]. However, utilizing encryption through the entire process can negatively affect total runtime due to the computational cost associated with homomorphic encryption.

III. Framework Design

We aim to ensure data privacy in distributed learning through a secure distributed learning framework. Our approach leverages homomorphic encryption to protect sensitive data. This section will cover our design goals, requirements, and the technology stack used.

A. Design Goals

Our framework addresses privacy concerns arising from distributed training. With data stored and processed outside a host's scope, data leakage is a significant risk, particularly in a crowd-sourced computing environment. We introduce a distributed training paradigm that utilizes homomorphic encryption to encrypt datasets at the host machine before distribution and training to mitigate this privacy concern. However, homomorphic encryption operations can be computationally expensive. Therefore, our algorithm employs a hybrid solution that reduces computational overhead at the cost of increased network traffic.

In addition to security, our design goals include usability, accuracy, and runtime performance. We aim to train models with accuracy comparable to traditional single-machine solutions while outperforming encrypted single-machine training solutions in runtime. To ensure security, no actor outside the data owner should have access to any plaintext part of the dataset.

B. Strategies for Distributed Learning

To develop a secure distributed training framework, we require a robust machine learning framework that supports granular changes to the training process to accommodate homomorphic encryption. While both PyTorch[1] and TensorFlow[2] are sound choices, we opt for TensorFlow due to its extensive support for distributed training.

Distributed learning requires special strategies, as regular machine learning algorithms do not work in this context. These strategies include:

- *Data parallelism*: In data parallelism, the dataset is divided into multiple parts, and each machine processes a different part simultaneously, with the gradients being averaged across all machines to update the model parameters.
- *Model parallelism*: Model parallelism involves processing different parts of the model on different machines, with each machine computing a portion of the model and sending the intermediate results to the next machine in the pipeline.
- *Federated learning*: Federated learning is a distributed learning strategy where the data remains on devices and is trained locally, while only model updates are sent to a central server for aggregation.

[1]https://pytorch.org/
[2]https://www.tensorflow.org/

In our specific use case, a single host owns both the data and model, rendering federated learning unsuitable. In addition, the usage of homomorphic encryption complicates model parallelism, thus making data parallelism the optimal option for our proposed framework. We implement data parallelism by dividing the encrypted dataset among participating workers and allowing them to work on a portion of the data. The server overseeing the workers receives the gradients computed by each worker, combines and averages them, and then dispatches model updates to each worker.

C. Homomorphic Encryption

Homomorphic encryption plays a vital role in our system design, allowing computations to be performed on encrypted data without the need for decryption. This feature enables workers to train on sensitive data without the risk of exposure. Homomorphic encryption was first proposed in 1978 by Rivest et al. [6] and has since undergone numerous advancements and iterations. Today, we have partially homomorphic encryption, somewhat-homomorphic encryption, and fully homomorphic encryption, with the latter being the most comprehensive.

The most commonly used fully homomorphic encryption (FHE) schemes for machine learning are BGV [24] and BFV [25], [26], which use bootstrapping to convert from somewhat-homomorphic encryption to fully homomorphic encryption. Recently, the CKKS [8] scheme has gained popularity due to its efficiency and ability to perform real number additions and multiplications, albeit with some approximation errors. However, AI models are designed to tolerate some errors in their computations, which is why we chose to use CKKS in our system.

We implemented CKKS using the TenSEAL library [27], which is based on Microsoft's SEAL library for homomorphic encryption [28]. TenSEAL supports various configurations and parameters that can affect the security and efficiency of the system. The two CKKS parameters we considered are the polynomial modulus degree and the coefficient modulus sizes. The polynomial modulus degree affects the size of ciphertext elements, the security level, and the computational overhead. The coefficient modulus sizes are a list of binary sizes used by TenSEAL to generate coefficient modulo primes. The length of this list determines the maximum number of rescaling operations performed on ciphertexts.

IV. System Specifications

Unlike traditional distributed learning systems, our proposed solution utilizes a hybrid structure where the host remains an active participant even after the data and model have been sent for training. Workers only use the encrypted dataset for the forward pass, resulting in an encrypted result vector that is sent back to the host. The host handles the decryption of the result vector, comparison to target values, and calculation of the loss function to prevent workers from seeing the plaintext labels for the dataset.

This approach reduces the computational toll homomorphic encryption takes on the worker, as only the forward pass requires homomorphic operations, allowing us to train AI models using the same encrypted inferences used in related papers. A high-level view of the dataflow between users and actors in our system is shown in Figure 1.

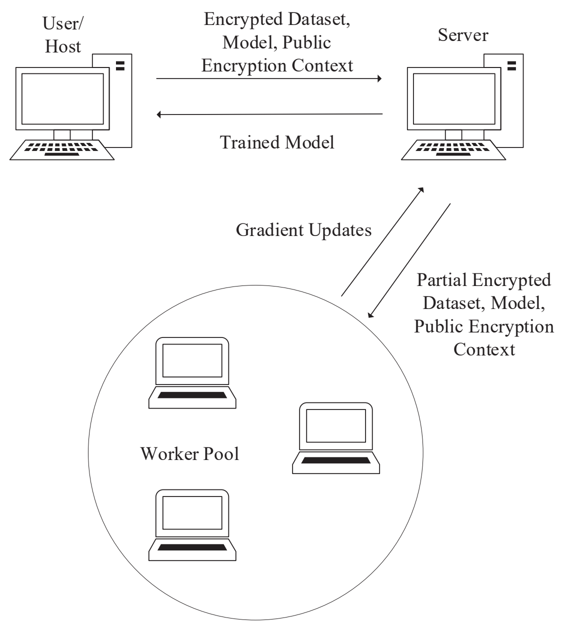

Fig. 1. Dataflow of the framework

A. Host (User)

A host user is a primary actor in our proposed framework who owns and seeks to protect their sensitive datasets. Host users may have several reasons for safeguarding their datasets, including complying with medical privacy laws and protecting personally identifiable information. Sometimes, a host user may not possess the computational resources needed to securely train their model in-house, necessitating the use of a secure framework like the one we propose.

Although the system handles training, a host is still responsible for data pre-processing, dataset encryption, and model parameter configuration. These processes follow the standard protocol for typical model training. Once the dataset and model are prepared, the dataset is encrypted using CKKS before being packaged with the model and sent to the server along with the public encryption context. After sending the encrypted dataset and model to the server, the host remains an active participant in the algorithm and responds to loss calculation requests from workers. These loss calculation requests are small and require a single homomorphic decryption per sample every epoch. The host compares the decrypted prediction with the expected result and generates a loss delta, which is sent back to the worker for backpropagation. Algorithm 1 shows the pseudocode for the algorithm run on the host.

B. Server (Worker Manager)

A server manages communication between the worker pool and host, directs the dataflow between each actor, and acts as the parameter server for the system. The server receives training packages from the host that contains an encrypted dataset, model, and public encryption context. It then repackages them into a smaller package based on the number of active workers. This new package is then sent to participating workers for training.

Algorithm 1 The Host Process

```
Require: Training dataset features $X$ and labels $Y$ with size
    $N$, Trainable AI model $M$, Batch size $B$, Epochs $E$,
    Activation function $F$
Ensure: Trained AI model $T$

  Preprocess dataset $X$
  Create encryption context $C$
  Create public context $C'$
  Encrypted dataset $X'$ = Encrypt $X$ with $C$
  Create training package $P = \{X', N, C', M, B, E\}$
  Connect to server
  Send $P$ to server
  while Training is not finished do
    Receive loss calculation package $R$ from server
    Calculate loss result package $L = F(Y, R)$
    Send $L$ to server
  end while
  Receive final model $T$ from server
```

The server stays active during training and is ready to accept both loss calculation requests and gradient updates from workers. It receives partial gradients from each worker, aggregates and combines them into a full model update, and sends the update back to each worker following every batch. Once all epochs are completed, the server sends the final model update back to the host. Algorithm 2 shows how the server operates.

C. Worker

The process executed by workers is outlined in Algorithm 3. Workers perform forward passes and backpropagation using encrypted data. Following each batch's forward pass, a worker sends a loss calculation request with the result vector to the host. The host calculates the loss and sends it back to the worker for backpropagation, preventing a malicious worker from performing a chosen ciphertext attack on the host. Using the GradientTape object in TensorFlow, workers calculate gradients for each batch with respect to the loss value. These gradients are then sent to the server, which aggregates and combines them into a full model update and sends the model update back to each worker following every batch.

V. Evaluation Plan

We conducted experiments to evaluate our proposed framework. This section will describe the test environment, dataset, training model architecture, CKKS parameters, and testing process used in our experiments.

Algorithm 2 The Server Process

```
Require: Number of workers $W$

Open communication thread for workers
Open communication thread for host
Connect to workers
Connect to host
Receive training package $P$ from host
Unpack $P = \{X', N, C', M, B, E\}$
for each connected worker do
    Create training package $K = \{X'/W, N/W, C', M, B, E\}$
    Send $K$ to worker
    while Training is not finished do
        Receive loss calculation package $R$ from worker
        Send $R$ to host
        Receive loss result package $L$ from host
        Send $L$ to worker
        Receive gradient update $G$ from workers
        Calculate model update $M'$ = Apply average $G$ to $M$
        Send $M'$ to worker
    end while
end for
Send final model $T$ to host
```

Algorithm 3 The Worker Process

```
Connect to server
Receive $K$ from server
Unpack $K = \{X'/W, N/W, C', M, B, E\}$
for each epoch $E$ do
    for each batch in $(X'/W)$ do
        Perform forward pass on $M$ with $B$ elements of $(X'/W)$ using $C'$
        Package result vector into loss calculation package $R$
        Send $R$ to server
        Receive $L$ from server
        Perform back-propagation on $M$ with $L$
        Calculate gradient update $G$
        Send $G$ to server
        if Training is not finished then
            Receive model update $M'$ from server
            Apply $M'$ to $M$
        end if
    end for
end for
```

A. Test Environment

We conducted all our tests on Google Colab[3], a cloud-based computing platform designed specifically for machine learning. We utilized separate instances of custom Google Compute Engine VMs for the host machine, server, and workers to emulate a distributed learning environment. These nodes were configured with the high-memory machine type, including 16 GB of RAM and two virtual CPUs. To simulate the use of separate physical machines as workers in our system, we deployed eight Google Colab instances as workers, each connected to a separate VM to ensure each worker had independent computing resources.

B. Dataset

We used the MNIST handwritten digit dataset [9], which consists of 60,000 training images and 10,000 test images, each measuring 28 $\times$ 28 pixels and labeled with one of the ten digits (0 to 9). We resized each image to 8 $\times$ 8 pixels using bicubic interpolation and normalized the samples using a mean of 0.1307 and a standard deviation of 0.3081. These 8 $\times$ 8 images are then flattened into a single vector of size 64 to fit the input shape of our model. We trained our model using only 10,000 samples from the training set, while the evaluation of the model used 1,000 samples from the test set. For reference, an example of a pre-processed sample before flattening can be seen in Figure 2.

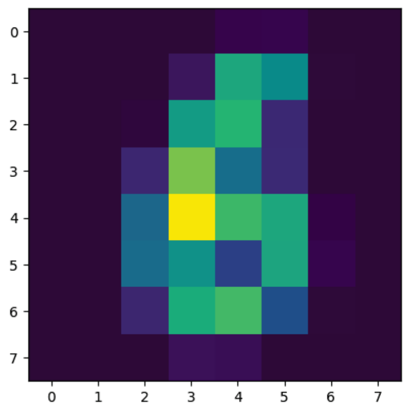

Fig. 2. A pre-processed sample from the dataset

C. Training Model Architecture

As shown in Figure 3, the model we used for our training is a fully connected neural network with an input shape of 64, two hidden layers with 32 and 16 neurons, respectively, and an output vector of 10, corresponding to the ten numeric digits we aimed to predict. This gave us 2,778 trainable weights in our model.

We utilized a square activation function for each hidden layer since homomorphic encryption can only support basic operations. For the loss function, we opted for cross-entropy

[3]https://colab.research.google.com/

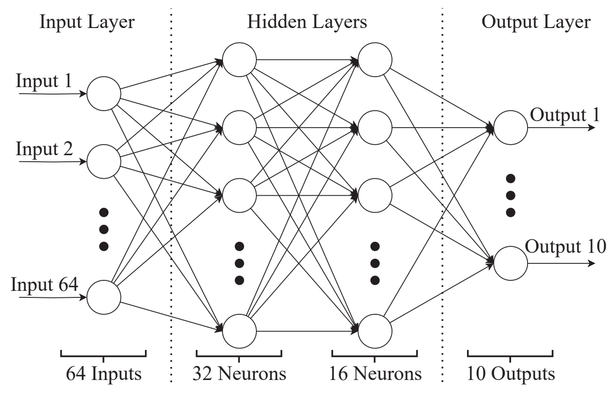

Fig. 3. Training model architecture

loss as it is best suited for categorization problems. We randomly initialized the weights and employed the Adam optimizer [29] with a learning rate of 0.001. With these parameters, our model performed three homomorphic matrix multiplications and six regular homomorphic multiplications per sample.

D. CKKS Parameters

The selection of CKKS parameters relies highly on the model architecture and the desired security level for the training process. We required an additional coefficient modulus element for each homomorphic multiplication operation in the model. In total, our model performed five total multiplications: three matrix multiplications from our dense layers and two multiplications from our square activation functions. Therefore, we choose the following parameters:

- *Polynomial modulus degree*: We set the polynomial modulus degree to 8,192 to accommodate the number of homomorphic operations performed in our test model.
- *Coefficient modulus sizes*: We used a 7-element list with a total of 192 bits to allow our test model to run its forward pass.

Regarding security, our chosen encryption parameters offer sufficient protection for the problem. As per the TenSEAL documentation [27], using a polynomial modulus degree of 8,192 along with a coefficient modulus size of 192 bits is sufficient for a 128-bit security level, which is deemed satisfactory by the National Institute of Standards and Technology [30].

E. Testing Process

After preparing our dataset and model, we proceeded to run them through our framework. The first step was for the host to create an encryption context using CKKS that was suitable for the model we intended to train. The next step was to encrypt the entire dataset using this context for training purposes. Finally, we serialized the encrypted dataset, model, and public encryption context and sent them to the server.

The server unpacked the serialized package and divided the encrypted dataset into N parts based on the number of active workers we used for training. The server then initialized an updated model, serialized it, and sent a training package to each worker.

Each worker unpacked their training package and began the training process. First, the worker ran a forward pass for each batch of samples, creating a batch of encrypted prediction results. The worker then serialized and sent this batch of predictions back through the server to the host, who decrypted the batch and calculated the loss for the entire batch. This loss was then sent back to the worker, who used it to calculate the gradient for the batch. The worker sent this gradient back to the server. After receiving all the gradients from each worker, the server combined them by taking the average. These combined gradients were then applied to the model, and the model update was sent to all workers for the next batch. This process continued until all batches were finished, resulting in a final model update after 10 epochs.

The final model update was sent to the host for evaluation and inference. We measured the framework's performance by timing each training run's total runtime, and the final model's accuracy was tested on a sample set of 1,000. We scaled the number of workers from 1 to 8 for these measurements.

VI. Evaluation Analysis

We conducted experiments to evaluate our proposed framework based on several key metrics, including total runtime, network delay, model accuracy, and security level.

A. Evaluation Results

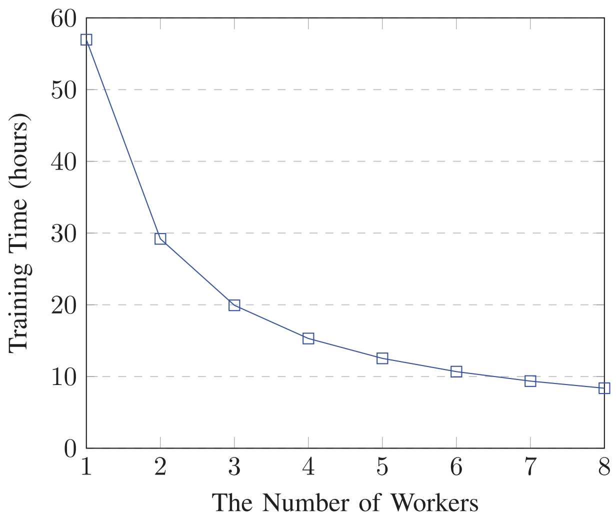

Fig. 4. Training time w.r.t. the number of workers.

We tested how long it took to train our framework by adjusting the number of workers involved in the process. Figure 4 shows the total time, including one-time operations such as dataset encryption and network overhead. Our framework significantly improved the training runtime as the number of workers scaled up. Using a single worker resulted in a training runtime of over 56.6 hours, with over 55 hours on forward passes alone. However, by utilizing more workers, we reduced the total training time to 8 hours, which is a reduction of over 80% of the total runtime. The computational impact on the host amounted to 63.3 minutes for encryption and decryption, after which the server and workers handled the rest.

We also assessed the overall network delay of our system. The overall network time was approximately 33 minutes in total across 10 epochs of training, with the majority of the time spent on transmitting the encrypted dataset. These 33 minutes represents 6.5% of the total training time of our fastest configuration of 8 workers. The encrypted dataset, when serialized, has a total size of over 4 gigabytes but only needs to be transmitted twice in full, with the full set going to the server and partial sets going to each worker. Loss calculation requests accounted for a much smaller portion of the network time as the encrypted results are much smaller than encrypted inputs. These runtime costs for the host and network remained fairly constant even as we increased the number of workers.

Figure 5 shows a graph of test accuracy versus the number of epochs trained. The accuracy of the completed model after 10 epochs evaluated on the test set comes to approximately 95%, indicating a strong prediction ability despite a precision loss from encryption. Our model converged to 95% by the 5th epoch, comparable to other state-of-the-art methods.

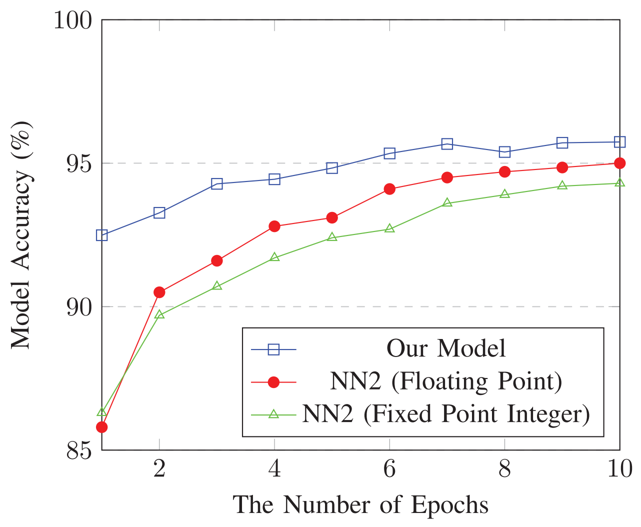

Fig. 5. Test model accuracy comparison.

B. Comparison to Prior Work

Our system outperforms the homomorphically encrypted system described in Nandakumar et al.'s work [10] in terms of training runtime while achieving a test accuracy of 95% after 10 epochs, which is comparable to their NN2 model. Nandakumar et al. reported a 40-minute training time for a mini-batch of 60 samples in their most optimized configuration. In contrast, our model trains the same-sized batch in just 3 minutes, thanks to our choice of encryption scheme and corresponding parameters.

Li et al.'s distributed learning framework [31] achieved a slightly higher classification accuracy of 96% for the MNIST dataset, with a faster convergence time of 300-400 seconds for their model. However, it is important to note that our model uses a fully connected neural network with the full set of labels from the MNIST dataset, while Li et al.'s work used a linear model and only two output labels. Additionally, our model took around 5 epochs to reach a 95% test accuracy, which is longer than Li et al.'s reported convergence time.

Jiang et al. utilized a homomorphically encrypted model called E2DM [32] to run inference on the MNIST dataset, which achieved an amortized inference time of 0.446 seconds per image and a classification accuracy of 98%. Our experiment prioritized faster runtime and stronger security level at the cost of some accuracy. We attribute the difference in model accuracy to their use of a pre-trained model and a convolutional neural network architecture, which is more suitable for image classification problems.

The Sphinx deep learning system [23] reports an inference time of 6 seconds in their evaluation results, but it includes the time taken to encrypt the sample. To make a fair comparison, we compared our amortized time against their batched training time, resulting in an amortized time of 0.78 seconds per sample. While our model's accuracy is slightly lower than Sphinx's, this difference could be attributed to their use of a convolutional neural network, whereas we employed a fully-connected neural network. In terms of communication overhead, Sphinx exhibits a lower communication cost per batch compared to our solution. Sphinx's communication cost is 18.3 MB per batch of 500 samples, while our solution requires approximately 235 MB. This discrepancy stems from a fundamental architectural difference. Our framework employs worker nodes in addition to the client and server nodes used by Sphinx. This enables our framework to parallelize the training process across multiple workers, thereby accelerating training time at the expense of communication time.

While some studies report a 99% accuracy rate for the MNIST dataset, such as Al Badawi et al. [17] and Ishiyama et al. [18], they used pre-trained models and focused on inference accuracy instead of training and validating a model from scratch. In the case of sufficiently accurate pre-trained models, the only accuracy loss from homomorphic encryption would be precision loss from the encryption algorithm itself.

Table I provides further comparisons of amortized forward pass/inference times, accuracy, and security level.

TABLE I
COMPARISON OF THIS WORK, NN2 [10], E2DM [32], AND SPHINX [23]

	Amortized inference time (s)	MNIST classification accuracy (%)	Security level (bits)
This work	0.265	95.7	128
NN2 [10]	40	95.0	80
E2DM [32]	0.446	98.0	80
Sphinx [23]	0.78	98.5	128

C. Significance

Our experiments demonstrate that implementing homomorphic encryption in a distributed learning setting can alleviate its impact on training time, despite incurring computational costs. Additionally, since most of the training process time is spent on performing homomorphic operations during the forward pass, our framework can efficiently scale up the number of workers before experiencing diminishing returns. These returns occur when the network load of managing dozens of workers surpasses the benefits of dividing the workload.

Furthermore, our results indicate that homomorphic encryption does not compromise model accuracy. Since real-life data often contains inherent noise, the loss of precision caused by encryption does not significantly affect the ability of the finished model to predict labels. However, it is crucial to set the correct parameters for the encryption context, as improper dataset pre-processing and analysis can affect overall security and training ability.

Our proposed framework represents a significant step towards a practical data privacy solution for small stakeholders who cannot compromise on security. By generating accurate models at the expense of runtime, our framework provides a service that brings homomorphic encryption much closer to practical use.

VII. Discussions

This research encountered several challenges, with the memory footprint utilized by the homomorphic encryption scheme being the most significant one. The serialized sample size increased from 64 bytes to over 400 kilobytes per sample, leading to a total encrypted dataset size of 4099 megabytes. While this is manageable for most high-end systems, it can lead to out-of-memory issues on moderately large datasets for hosts with limited hardware or workers with low-end specifications. To resolve this issue, future work can focus on reducing the memory footprint of the homomorphic encryption scheme or using physical storage in the design to trade speed for reliability.

Load balancing will become a more significant issue as the framework scales up. Our test utilized a fairly uniform distribution of machines with similar, if not identical, specifications. However, in a production environment, workers are not necessarily equal in computing power, which can cause delays as the system waits for slower workers to catch up. Load-balancing techniques that seek to alleviate this issue are available today, but it is challenging to apply these without significant changes due to the size of the training packages. One potential solution is to have each worker report their computing power to the server, which then calculates the proper workload distribution to ensure every worker finishes at a similar time. This line of work can be used to expand the framework into a crowd-sourced distributed learning platform.

When using homomorphic encryption, the amount of time it takes to complete encryption operations is the biggest factor in runtime. Our research shows that the time it takes to complete inference is equal to the time it takes to complete network operations when using 100 workers. After this point, the time it takes to send each worker their training package becomes longer than the time it takes to train on it. Future work can explore ways to optimize the computational overhead of homomorphic encryption operations. Another way to reduce the overall training time is to increase the speed of the network connection.

Dataset encryption at the host becomes an increasingly significant part of the total runtime as the number of workers increases. The one-time cost of encrypting the entire 10,000-sample dataset was an hour in our tests. At the highest test configuration of 8 workers, this encryption time accounted for 12% of the total runtime. However, dataset encryption only needs to occur once for a given training problem and can be reused for training multiple models.

The use of the CKKS scheme introduces a vulnerability that can lead to a passive attack and the recovery of the secret key, as described by Li and Micciancio [33]. Cheon et al. proposed a fix in the form of a decryption-for-sharing operation for the CKKS scheme that makes sharing the decrypted result safe [34]. However, this approach can expose dataset labels if used in plain by workers during loss calculation. To solve this problem, we present a novel solution that shifts the responsibility of loss calculation from the worker to the host. After a forward pass, the worker sends the encrypted result vectors to the host, who decrypts the result and calculates the loss using the loss function on the host side. The calculated loss is then sent back to the worker for backpropagation. By adopting this approach, we prevent the leakage of raw result values and thwart chosen ciphertext attacks from malicious workers by obscuring the values sent back to the worker behind an undisclosed loss function. Both the server and workers only have access to encrypted result vectors and loss values, which do not reveal information about the original dataset. The security of our framework is based on the strength of the encryption algorithm. CKKS utilizes ring learning with errors as its encryption algorithm, which is a type of lattice-based cryptographic system. With a security level of 128 bits, CKKS is considered quantum-safe against the current computational capabilities of today's machines.

VIII. Conclusion

As machine learning becomes increasingly prevalent, data privacy and security concerns are becoming more significant. Our distributed learning framework, which incorporates homomorphic encryption, provides a potential solution to these concerns. Our experimental evaluation demonstrates that homomorphic encryption can be used to secure data in a distributed learning framework with minimal impact on training times compared to using homomorphic encryption alone. Furthermore, our framework's architecture does not hinder the ability to train an accurate AI model, and we have found that our model converges to a solution in fewer epochs than other secure training platforms.

However, the repeated homomorphic operations in the training process can lead to reduced usability, as our solution takes significantly longer to perform the same training process than unencrypted training.

Future work can focus on expanding the framework's security to cover data integrity, availability, or authenticity and detecting and preventing model poisoning. In addition, improvements to the homomorphic encryption scheme can significantly enhance the model's performance, especially in the inference stage, by utilizing batch processing.

In conclusion, our framework provides an efficient way to use homomorphic encryption in machine learning while preserving data privacy and security. It presents a promising solution for organizations seeking to use sensitive data for machine learning while adhering to data security requirements.

REFERENCES

[1] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2020.

[2] Ponemon Institute LLC, "Data protection and privacy compliance in the cloud: Privacy concerns are not slowing the adoption of cloud services, but challenges remain," https://azure.microsoft.com/mediahandler/files/resourcefiles/ponemon-privacy-cloud-research/Ponemon-privacy-cloud-research.pdf, 2020, (accessed Apr. 11, 2023).

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

[4] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 308–318.

[5] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*. IEEE, 1986, pp. 162–167.

[6] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of Secure Computation*, vol. 4, no. 11, pp. 169–180, 1978.

[7] B. Pulido-Gaytan, A. Tchernykh, J. M. Cortés-Mendoza, M. Babenko, G. Radchenko, A. Avetisyan, and A. Y. Drozdov, "Privacy-preserving neural networks with homomorphic encryption: Challenges and opportunities," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1666–1691, 2021.

[8] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, June 2019.

[11] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[12] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[13] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 2020)*, 2020.

[14] Y. Wang, M. Gu, J. Ma, and Q. Jin, "DNN-DP: Differential privacy enabled deep neural network learning framework for sensitive crowdsourcing data," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 1, pp. 215–224, 2019.

[15] R. Hu, Y. Guo, H. Li, Q. Pei, and Y. Gong, "Personalized federated learning with differential privacy," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9530–9539, 2020.

[16] J. Domingo-Ferrer, D. Sánchez, and A. Blanco-Justicia, "The limits of differential privacy (and its misuse in data release and machine learning)," *Communications of the ACM*, vol. 64, no. 7, pp. 33–35, 2021.

[17] A. Al Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1330–1343, 2020.

[18] T. Ishiyama, T. Suzuki, and H. Yamana, "Highly accurate CNN inference using approximate activation functions over homomorphic encryption," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 3989–3995.

[19] S. Meftah, B. H. M. Tan, C. F. Mun, K. M. M. Aung, B. Veeravalli, and V. Chandrasekhar, "DOReN: Toward efficient deep convolutional neural networks with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3740–3752, 2021.

[20] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, no. 3-4, pp. 121–136, 1975.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[22] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[23] H. Tian, C. Zeng, Z. Ren, D. Chai, J. Zhang, K. Chen, and Q. Yang, "Sphinx: Enabling privacy-preserving online learning over the cloud," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2487–2501.

[24] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[25] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Springer, 2012, pp. 868–886.

[26] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[27] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "TenSEAL: A library for encrypted tensor operations using homomorphic encryption," *arXiv preprint arXiv:2104.03152*, 2021.

[28] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-SEAL v2. 1," in *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 2017, pp. 3–18.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] E. Barker and Q. Dang, "NIST special publication 800-57 part 1, revision 4," *NIST, Tech. Rep*, vol. 16, 2016.

[31] J. Li and H. Huang, "Faster secure data mining via distributed homomorphic encryption," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2706–2714.

[32] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1209–1222.

[33] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*. Springer, 2021, pp. 648–677.

[34] J. H. Cheon, S. Hong, and D. Kim, "Remark on the security of CKKS scheme in practice," *Cryptology ePrint Archive*, 2020.