Effect of Homomorphic Encryption on the Performance of Training Federated Learning Generative Adversarial Networks

Ignjat Pejic
Supervisor(s): Kaitai Liang, Rui Wang
EEMCS, Delft University of Technology, The Netherlands
22-6-2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

A Generative Adversarial Network (GAN) is a deep-learning generative model in the field of Machine Learning (ML) that involves training two Neural Networks (NN) using a sizable data set. In certain fields, such as medicine, the data involved in training may be hospital patient records that are stored across different hospitals. The classic centralized implementation would involve sending the data to a centralized server where the model would be trained. However, that would involve breaching the privacy and confidentiality of the patients and their data, and would be unacceptable. Therefore, Federated Learning (FL), a ML technique that trains ML models in a distributed setting without data every leaving the host device, would be a better alternative to the centralized option. In this ML technique, only parameters and certain metadata would be communicated. In spite of that, there still exist attacks that can infer user data using the parameters and metadata. A fully privacy preserving solution involves homomorphically encrypting (HE) the data communicated. This paper will focus on the performance loss of training a FL-GAN with three different types of homomorphic encryption: Partial Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE). We will also test the performance loss of Multi Party Computations (MPC), as it has homomorphic properties. The performance will be compared to the performance of training an FL-GAN without encryption. Our experiments show that the more complex the encryption method is, the longer it takes, with the extra time taken for HE being quite significant in comparison to the base case of FL.

Keywords: *Generative Adversarial Network, Federated Learning, Privacy Preserving, Homomorphic Encryption, Neural Network*

1 Introduction

Machine learning (ML) is a subset of artificial intelligence (AI) that allows models to learn from the data provided to them. One such model is a Generative Adversarial Network (GAN) [7], which has positively impacted the field of generating fake data that appears to be real. To accurately train the GAN, a considerable amount of data may need to be used, and that data is not always centralized. Therefore to obtain the data, users need to willingly share their data, which is not always the case due to privacy concerns. To avoid such issues, Federated Learning (FL) [16] is a better training technique for GANs. FL is a machine learning model that allows GANs to be trained in a collaborative and distributed fashion, where the data does not have to be shared or communicated outside of the model training facility. The information shared will only consist of the parameters of the locally trained models and metadata, which is a huge improvement from the initial approach. Unfortunately, according to [22], there is still a risk that malicious users or the main server may be able to infer the data from the parameters and metadata. The malicious user may recover data samples by solving for the optimal pair of inputs and outputs to match the parameters communicated to the main server [20]. To address this issue, one privacy preserving implementation involves using Homomorphic Encryption (HE) on any data that is to be communicated [22]. This will enable us to train any model that requires sensitive data more accurately as we will be able to obtain more data to train them.

To illustrate the importance of using HE with FL-GANs, we will provide an example. If we wanted to study different medical images, we could use HE with FL-GANs, to train a Generator that is able to produce realistic fake images. We can then create numerous instances of certain types of diagnoses, and be able to further improve our doctors' abilities to study and understand the medical images. That can save human lives, and may not be possible if patients would be unwilling to share their data for fear of privacy invasion.

While using HE on FL-GANs solves the issue of privacy, there is a significant performance loss. This is where our research topic comes in. The topic is as follows:

- Implement a Federated Learning Generative Adversarial Network with Homomorphic Encryption, and study the effect of HE on the performance (and accuracy) of training the model.

We will implement the FL-GAN and study the performance with three different types of HE: Partial Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE) [12]. We will also use a tool implementing Secure Multiparty Computation (MPC) [19], that has both additive and multiplicative homomorphic properties. Each of these types of encryption (and MPC) will be explained in further detail in the section 2. This paper will not involve proving whether or not the encryption schemes ensures privacy, as it has already been proven that they do [21].

This paper will be organized as follows. Section 2 will contain the background information, where GANs, FL,HE, and MPC will be explained and discussed in further detail. Then, section 3 will go over our System Model, describing the threat model and FL algorithm. Moving on, section 4 will elaborate on our research approach and implementation, where the experiment and tools used will be described. We will go over the implementation of the topics discussed in section 2 and how they fit together to get our desired result. After that, section 5 will go over the results that our experiment has obtained on the performance loss with HE and MPC. Moving forward, in section 6 we will discuss our results and go over them while also explaining why anything is missing. We will mention any unexpected results, and try to elaborate on the possible reasons for them. The following section, section 7, will be about responsible research. We will discuss how this research has been conducted in an ethical and moral way, and what we did to obtain the most accurate results. In section 8, we will go over the possible future work and research that should be done to increase our understanding of the topic. It

will be based on our ideas developed through researching our topic. Lastly, in section 9 we will conclude our research by providing a summary of what was done.

2 Background Knowledge

The coming subsections will go over the background knowledge that we deemed essential to understand all parts of the report. We will discuss the GAN, FL, HE, and MPC.

2.1 Generative Adversarial Network

A GAN is a generative machine learning model consisting of two Neural Networks that compete against each other with the goal of being able to generate realistic fake data [13]. The two models in question are Discriminators and Generators. A Discriminator is, at its basic form, a classifier used to distinguish between real and fake data. On the other hand, a Generator is a model, that when provided with a random input (random noise), generates fake data [**?**]. The process of training a GAN starts with training the Discriminator for a certain number of epochs (number of iterations of data set) on a labelled data set, followed by training the Generator on the Discriminator for also a number of epochs. The training is done using the minimax equation, stated below:

$$\min_G \max_D E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[1 - \log D(G(z))] \tag{1}$$

What the above equation states is that the Discriminator tries to maximize the likelihood of determining the correct label (real or fake) for both the training examples and for examples generated by the Generator. On the other hand, the Generator tries to minimize the probability that the Discriminator can correctly classify its generated images.

This whole process gets repeated multiple times. The training process is ideally completed when the Discriminator can predict whether generated data is real with a probability of 50%. As that is practically difficult to achieve, the usual way of training a GAN involves repeatedly training the Generator and Discriminator for a certain number of epochs, and then measuring the accuracy of the generated images.

2.2 Federated Learning

The way a machine learning model is trained can have a huge impact on the privacy of the model. Initially, most ML models were trained in a centralized setting, meaning all of the data had to be sent by the data owners to a main server, where it would be used to train the model. However, in a scenario that involves hospital patient data, for instance, the data owners may be unwilling to share the data due to privacy concerns. In order to maintain data confidentiality, a technique called Federated Learning was developed [24].

While there do exist different forms of Federated Learning, the one we will use in this research paper will be a centralized FL using the fed-avg algorithm [15]. This involves one main central server and multiple client nodes. Each of the client nodes will have its own ML model with randomly initialized model parameters, where it will be trained using the local data. As the training process takes place, the model parameters will be updated for each. Once the clients have trained their models, each client will send its model's parameters to the centralized server. The server aggregates the parameters, and sends them back to the individual client nodes. The training process then repeats until the models are adequately trained.

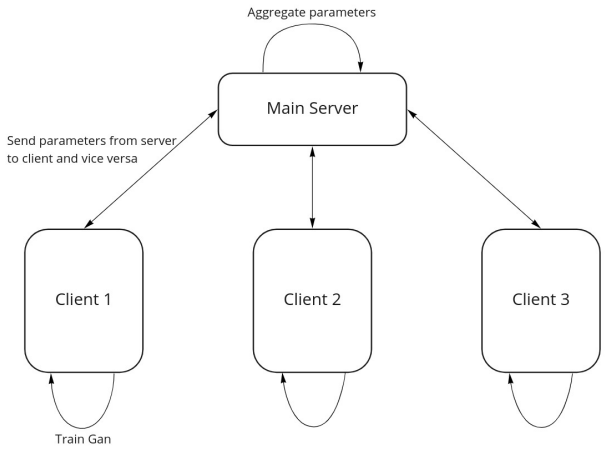

Figure 1: Visualization of FL-GAN

An example of an FL-GAN can be seen in figure 1.

2.3 Homomorphic Encryption

HE is a technique used to protect data confidentiality of users [26]. While standard encryption (non HE) is used to make the data unreadable, it does not allow for any computation to be performed on the data. In order to do any computations, the data would first have to be decrypted, which is a privacy risk. On the other hand, data encrypted with HE may be updated with the use of arithmetic computations, which makes it desirable for increasing the privacy in FL and cloud storage/computation.

The formula for basic Homomorphic Encyption is the following:

$$c_1 = E_k(x), c_2 = E_k(y) \tag{2}$$

$$D_k(c_1 + c_2) = x + y \tag{3}$$

$$D_k(c_1 * c_2) = x * y \tag{4}$$

As mentioned beforehand, there are three different types of HE: namely Partial Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE).

- PHE allows the user to perform either addition or multiplication (equation 3 or 4) on the encrypted data. In this report, PHE will be assumed to be additive (equation 3).
- SHE is a more powerful form of encryption, which allows one to perform both additive and multiplicative operations on the ciphertexts (equations 3 and 4). However, there are only a certain amount of operations that can take place. Any operations past this point will not guarantee that our decryption provides the correct result. It is slower than PHE.

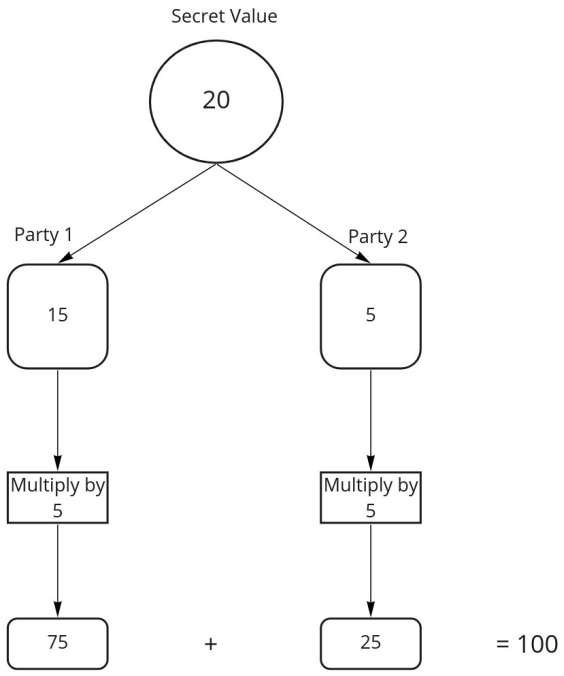

Figure 2: Example of MPC

- FHE is the most powerful form of encryption, but also the slowest. Like the SHE, it allows for both addition and multiplication to be done on encrypted data (equations 3 and 4). The difference is that with FHE there can be an unlimited amount of arithmetic operations that are allowed to take place, using a technique called bootstrapping.

2.4 Secure Multi Party Computation

While the main topic of this paper is about the performance loss in training FL-GANs using HE, we also thought it would be interesting to add a tool for comparison that has both additive and multiplicative homomorphic properties (MPC). MPC is a subfield of cryptography, where performing a computation works by distributing it across multiple parties. Each party may only see its own data. This, in combination with the fact that no party further shares its own data, allows for privacy. It has a higher performance than Homomorphic Encryption.

Figure 2 shows an example of MPC.

3 System Model

This section will go over the proposed implementation of our Federated Learning Generative Adversarial Network with Homomorphic Encryption. We will first go over the threat model, before providing our algorithm of the ML model.

3.1 Threat Model

There are multiple threats that can potentially affect our Federated Learning system, namely the clients that train the GANs, the server that aggregates the parameters received from the clients, and any potential outsiders. We assume our key generation is secure, and its communication to the clients is secure. Moreover, we will consider that both the clients and the server are following the honest-but-curious model, meaning that none will deviate from the algorithm that exists, but will try and learn as much as possible from any received information. When it comes to the potential outsiders, we assume that they may intercept any message except the key generation and its communication.

Table 1: Notation Table

Notation	Definition
n	Number of clients
pd	Parameters of Discriminator
pg	Parameters of Generator
dd	Dataset
s	Sum of parameters
rp	Parameters received from server
$\emptyset$	Empty set

3.2 Experiment Algorithm

Our algorithm will consist of a centralized server, that will send each partition of the main dataset to its corrsponding client node. The client node will then train its individual GAN on the received dataset, before encrypting the parameters and sending them back to the server. The server will then use the fed-avg algorithm [25] to aggregate the results. This will, in the simplest terms, involve averaging the parameters of all clients, before sending them back to each client. The clients will then decrypt the received parameters, update the model, and keep training the model for as many epochs as necessary.

For the client side algorithm, refer to Algorithm 1, while for the server side use Algorithm 2. Make sure to use table 1 for all the notations to understand the algorithms.

At the beginning of the process, each public and private key is generated by the key generator and then sent to each of the clients. As mentioned previously, we assume that the communication of the keys is fully secure. The key generators no longer participate in any part of our machine learning model and receive no data whatsoever.

4 Research Approach and Experiment Setup

This section will discuss the overall approach to begin answering the research question, alongside all of the considerations that needed to be taken into account. This section will dive more into the implementation details in comparison to the algorithm of section 3.

4.1 Research Approach

Initially, after going through the existing research on FL, GANs, and HE, a number of observations were made. There exists a lot of research on each topic individually, on the GANs trained with FL, and on the effect of HE on FL. However, we have been unable to find research encompassing all three fields simultaneously.

Algorithm 1 Algorithm 1: GAN training on client

Require: pub_key, $priv\_key$, $dataset$
  $d \leftarrow discriminator()$
  $g \leftarrow generator()$
  while $dd \neq \emptyset$ **do**
    train(d, g)
  end while
  for p in pg **do**
    $p \leftarrow p/n$
    $p \leftarrow Enc(p)$
    send encrypted parameter to server
    receive aggregated parameter from server
    $p \leftarrow Dec(p)$
  end for
  for p in pd **do**
    $p \leftarrow p/n$
    $p \leftarrow Enc(p)$
    send encrypted parameter to server
    receive aggregated parameter from server
    $p \leftarrow Dec(p)$
  end for

Algorithm 2 Algorithm 2: Server

  $dd \leftarrow importdataset$
  for c in n **do**
    $partition\_c \leftarrow dd/n$
    send $partition\_c$ to $client\_c$
  end for
  Wait for clients to train GAN
  for p in pg **do**
    $s \leftarrow \emptyset$
    for c in $number\_of\_clients$ **do**
      $s \leftarrow s + rp$
    end for
    send parameters back to clients
  end for
  for p in pd **do**
    $s \leftarrow \emptyset$
    for c in n **do**
      $s \leftarrow s + rp$
    end for
    send parameters back to clients
  end for

What was also unfortunate is that the majority of the research papers did not have the code base used to conduct their experiments shared publicly, making it challenging to recreate their results or further improve on their own. Moreover, for the repositories that were publicly shared, a lot of documentation was either missing or incorrect, leading to the same issues as stated above.

Due to the reasons stated in the previous paragraph, it became clear that we would need to integrate the three components ourselves. Given the time constraints of the projects, we decided that the best approach was to look at industry standard tools used for secure machine learning, as we expected them to have the best documentation and support of use. What we have not been able to implement will be found in section 8 as the future work.

4.2 Experiment

As stated in the Research Approach subsection, we decided to use the most used industry standard tools for our integration of GANs, FL, and HE. We will go over the implementation process, starting with unsuccessful attempts before moving on to our approach that turned out successful.

To begin with, there are two main open source repositories used for Machine Learning, namely PyTorch [23], developed by Meta/Facebook, and TensorFlow [1], developed by Google. Using these softwares, it is possible to create most of the standard ML models. There exist many implementations of GANs in both, allowing us to easily construct and train our own model. When it comes to training those models using FL, there exist a number of implementations including preexisting frameworks: PySyft [27] for PyTorch and TensorFlowFederated for TensorFlow [2]. The final step would be to be able to use HE alongside such models. This is where the main issue appeared.

Both PyTorch and TensorFlow at one point had their own frameworks for homomorphic encryption: PySyft for PyTorch again and TF-Encrypted for TensorFlow. However, as of May 2022, both of the frameworks are unusable for this project. Starting with PySyft, the software has recently been and still is being updated. While the software is being updated, the documentation is still the same. While there do exist many examples of HE with PyTorch in the past, they do not work with the new version of PySyft. Moving on to TF-Encrypted, the repository has not been updated recently. Therefore, it is currently incompatible with the newest version of TensorFlow. As a consequence of neither framework allowing for direct HE, we have decided to work on creating an implementation using PyTorch, as we found it easier to use and modify to our advantage.

We started out by implementing a FL-GAN using the Message Passing Interface (MPI) [8], which was initially created by a group member. It allows for parallel computing and has been tried and tested. For our experiment, we decided to initially use four nodes, consisting of one server node and three client nodes. The Neural Networks of the GAN, the Discriminator and Generator, are trained on each client node. The data used for training the GAN is the CIFAR-10 dataset, which consists of 60000 32x32 color images of 10 categories, namely airplanes, cars, birds, cats, deer, dogs, frogs, horses,

ships, and trucks. The 60000 images are split even among the categories.

After the models are trained, their parameters are sent to the main server node via a data structure called a tensor. The tensor has a similar structure to a n-dimensional matrix, but is specific to PyTorch. The default datatype of each element is a float32. At this stage, the server aggregates the data using an algorithm called fed-avg. This technique involves taking the mean of each parameter. After the aggregation takes place, the data is sent back to each client for the training process to continue. This process is repeated for as many epochs as desired.

At this point we had an implementation of the FL-GAN, but we were still missing HE. In order to include it, the next step was to encrypt all parameters on the client side before they were communicated with the server. Where possible in our implementation, we tried to avoid giving our server the encryption keys.

Unfortunately, two issues were present. Firstly, due to the implementation of MPI, it was not trivial to send the keys from the key generator to the clients. Namely, certain methods of implementation required encryption contexts. Sending those contexts using MPI is not possible, as that is one of the things that MPI forbids to be sent and received. Moreover, certain forms of encryption slowed down the main process by a large amount, making it unfeasible to run on our machine.

Due to the concerns above, we have instead decided to approximate the extra time taken to train the FL-GAN with HE. To achieve this, we have recorded the time taken to generate all of the necessary encryption keys. Afterwards, we recorded the time taken to both encrypt and decrypt each value. Using these calculations and our knowledge of the total number of parameters involved in training the GAN, we approximated the time taken for all the necessary encryptions and decryptions. Moreover, where necessary, we measured the difference in size between plaintexts and encrypted ciphertexts, and used that to approximate the extra time taken for communication between the clients and the server. The main algorithm used for our entire implementation can be found in section 3.

5 Results

After implementing our model, we have recorded/approximated the performance loss of training FL-GANs with HE. The first subsection will discuss the results of running the model without any HE, and each following subsection will present the additional performance loss for a specific type of HE or MPC. Some terms used in this section include t, the number of tensors, p, the total parameter data found in all of the tensors for both the Generator and Discriminator, c, the number of clients, and e, the number of (federated) epochs. In our experiments, t is 13 for the Generator and 11 for Discriminator, p is 6342272, c is 3, and e is 50.

All the tests were implemented in Python 3.7, and were performed on a machine with a Intel (R) Core (TM) i7-8750H processor with 2.20GHz. Due to the way MPI works, the number of cores used depends on the number of clients. There is an additional core for the main server and one for a key generator whenever necessary.

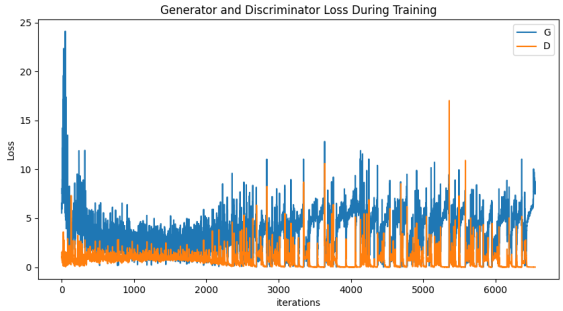

Figure 3: BCELoss for Generator and Discriminator as the number of training iterations increases

5.1 Base Case - FL-GAN With No HE

As mentioned in sections 3 and 4, we have run our implementation of the FL-GAN using three clients and a central server, aggregating the parameters using an algorithm called fed-avg. We have run the federated algorithm for a total of 50 epochs, with each of the three clients using a third of the CIFAR-10 dataset for training the GAN. Their individual dataset remains constant for the entirety of the training process.

After our implementation has finished running, we have collected our results. We have only run the process three times, due to the length of time necessary. The average training process took a total of 16 hours, and we measured the accuracy using Binary Cross Entropy Loss (BCELoss). BCELoss is the loss observed in binary classification. It increases for the Discriminator when it incorrectly classifies a real or generated image, and increases for the Generator if the Discriminator believes that the generated image is fake.

5.2 FL-GAN with PHE

In this subsection, we have used an algorithm for Partial Homomorphic Encryption, namely the Paillier Cryptosystem (PC) [4]. This algorithm enables us to do add any ciphertexts or multiply them by a constant value. The tool we used as our PC is [9], as our research showed that this library is used and referenced the most in the Python language.

In our implementation, the worst case time complexity for adding the PC is $\mathcal{O}(p)$. Ideally, we would be able to encrypt each tensor, which is expected to take less time and be more optimized. Unfortunately, given that those tools are being updated, we had to iterate over each parameter in each tensor and encrypt each of those values.

We have decided to compare the performance loss of adding the PC using 4 different keys, namely: 64 bits, 128 bits, 256 bits, and 512 bits. To approximate the extra time needed, we have recorded the time taken to first generate the keys. This was less than 0.1 seconds, making it negligible. Then, the time taken to encrypt and decrypt one value was recorded. For both of these recordings, an average time was taken. The average of a minimum of 10000 runs was done for each recording. We then had to multiply the time taken

Table 2: Results for Paillier Cryptosystem for 3 clients and 50 epochs

K.S. [a]	T.F.O.E. (s) [b]	T.F.O.D. (s) [c]	T.A.T.P.C.P.E. (s) [d]	T.A.T.T. (s) [e]
64	0.000105	1.612e-5	771	115650
128	0.0005040	0.00018	4311	646650
256	0.00134	0.00058	12177	1826574
512	0.00671	0.00023	44015	6602305

[a] Key Size
[b] Time for one encryption
[c] Time for one decryption
[d] Total additional time per client per epoch
[e] Total additional time taken (3 clients,50 epoch)

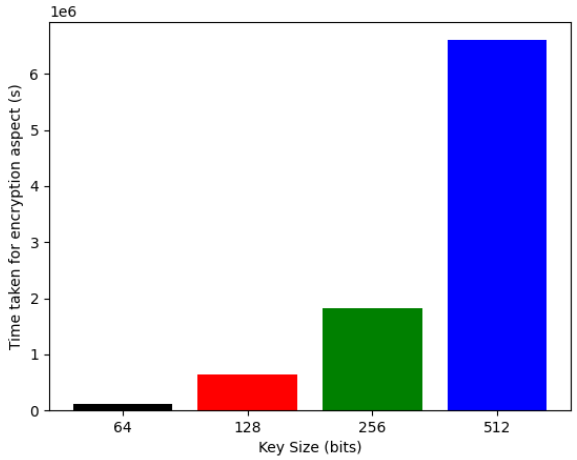

Figure 4: Total time lost due to PHE depending on key size

for one encryption and decryption by p, to get the total encryption and decryption time for one client for one training epoch. Lastly, we multiplied that by c and then by e. We have also recorded the size of the ciphertext for each key length. It is worthwhile mentioning that while the ciphertexts were of different size, the Paillier object that was stored and communicated was always 64 bits. Due to some implementation issues, we decided that instead of communicating tensors, we would first convert them to numpy arrays, encrypt the values, and then communicate them for aggregation. The time taken for converting between a tensor and a numpy array was negligible (less than 2 seconds total). Moreover, the aggregation also did not add up to any significant time (also less than 2 seconds).

All of the other results can be found in table 2. They will include the total time lost depending on the key size. As our results only contain the performance loss for 3 clients and 50 epochs, equation 5 will state the general formula for the total time lost.

$$total\ time\ lost = p*(encryption+decryption\ time)*c*e \quad (5)$$

Figure 4 visually presents the total time lost for each of the four key sizes. Lastly, as the PC has no losses with encryption and decryption, there is no accuracy loss in training the GAN in comparison to the base case.

Table 3: Results for SHE for 3 clients and 50 epochs

A [a]	E.T.P.I.P. (s) [b]	E.T.T.I.T. (s) [c]	D.T.P.I.P. (s) [d]	D.T.T.I.T. (s) [e]	T.P.C.P.E. (s) [f]	TT (s) [g]
1	0.00397	N/A	0.00113	N/A	32322	4848408
2	N/A	0.00487	N/A	0.140 - 0.377	14.25	2137

[a] Approach
[b] Encryption time p in p
[c] Encryption time per t in t
[d] Decryption time p in p
[e] Decryption time t in t
[f] Time per client per epoch
[g] Total time

5.3 FL-GAN with SHE

When it comes to SHE, we have come across two main schemes. The first is the Brakerski-Fan-Vercauteren (BFV) scheme [14], which is used when working with integers. The second scheme is the Cheon-Kim-Kim-Song (CKKS) scheme [5], which allows for encryption and decryption of real numbers. Both of these schemes are layered, meaning that we there is a limit on the number of arithmetic operations we can make on the ciphertext before it becomes undecipherable. As our tensor stores values which are float32, we have decided to go with the CKKS scheme.

As with all other SHE implementations, we can do both addition and multiplication between cipher texts. Not only that, as CKKS is based on Learning With Error (LWE), it is quantum secure. We were fortunate to find an implementation of CKKS that works with our PyTorch implementation: TenSEAL [3]. TenSEAL is built on top of Microsoft SEAL, the industry standard to both SHE and FHE.

When it comes to predicting the total time loss, we have decided to go with two different approaches. The first is similar to what we did with the PC, where we encrypted every parameter of every tensor. This also has a time complexity of $\mathcal{O}(p)$. However, due to the fact that TenSEAL is compatible with PyTorch, we were able to encrypt tensors as a whole, improving our time complexity to $\mathcal{O}(t)$. The first approach was done to have a suitable comparison with the PC. We are only able to use a 128 bit key with TenSEAL, so that was our key strength for both of the approaches. The performance based results for both of the approaches can be found in table 3.

For approach one, we can also use equation 5 to predict the total loss for any number of clients and epochs. For approach 2, we can refer to equation 7.

$$tt = total\ tensor\ encryption\ and\ decryption\ time \quad (6)$$

$$total\ time\ lost = tt * c * e \quad (7)$$

Unfortunately we do not only have a time based performance loss. CKKS supports approximate arithmetic calculations over real numbers, meaning that our operations will have some losses in accuracy, affecting our entire model. The reasoning is that at a certain point in the CKKS encryption scheme, we need to round our transformed ciphertext. This rounding will cause a loss. More on this can be found here [6].

As mentioned in section 4.2, MPI is unable to communicate everything, including contexts. As TenSEAL uses contexts, we are unable to communicate both the keys and ciphertexts, not allowing us to present any performance metric.

Table 4: Results for MPC for 3 clients and 50 epochs

TET (s) [a]	TDT (s) [b]	TTPC (s) [c]	TT (s) [d]
0.000184 - 0.0774	0.000153 - 0.03821	0.368	55.2

[a] Tensor Encryption Time
[b] Tensor Decryption Time
[c] Total time per client
[d] Total time

5.4 FL-GAN with FHE

Regarding FHE, we do not have any results due to time constraints. We were planning to run the same CKKS scheme that we had mentioned in the previous subsection, but would add bootstrapping to the system. We again would have had to use only the CKKS scheme, due to the data type. We predict that this would have taken a lot of extra time. This will further be mentioned in the discussions sections 6.2 and 8.

5.5 FL-GAN with MPC

Similarly to TenSEAL, we were able to find a software repository that used Secure Multi Party Computation that was compatible with PyTorch. The tool in question is Crypten [18]. Using Crypten, we were able to encrypt whole tensors at a time, making the time complexity $\mathcal{O}(t)$. While the previous subsections had an approach encrypting every parameter of every tensor, we decided that this was unnecessary for MPC as it is not HE, making an exact comparison between two methods of lower priority. The results for this approach can be found in table 4. As with the second approach of SHE, we can use equation 7 to estimate the time loss for any number of clients and epochs.

This approach also has accuracy losses, with each parameter in each tensor being off by 2.2798291e-05. This difference, over 6342272 parameters will add up to an accuracy loss.

6 Discussion

This section will discuss two topics. We will first go over our results, and then mention what results are missing and we will explain why so.

6.1 Result Analysis

Overall, most of the results that we got were as expected. When it comes to the additional time increase, we expected Multi Party Computations to be the quickest, followed by Partial Homomorphic Encryption, and lastly Somewhat Homomorphic Encryption.

The base case scenario of running our FL-GAN on the CIFAR-10 dataset with 3 clients ran in 16 hours. The images generated were similar to [17], a repository use to train a normal GAN on the CIFAR10 dataset. The images produced show progress as the training process continues for the most part, but as Figure 3 shows at a certain point the BCELoss stops decreasing, and then starts variating inconsistently. This was unexpected, but we believe the reason could be due to the fact that we over trained the Discriminator. We can see in the figure that in many instance were the Discriminator loss is near 0, the Generator loss increases. As the Discriminator gets better and better trained, it can become more challenging for the Generator to be trained properly, as almost anything not perfect may be rejected by the Discriminator.

When it comes to the PC, the stronger the security was the longer it took to encrypt all the necessary parameters. The total time increase between the 64 bit key and the 512 bit key was from just above 32 hours to just under 1834 hours. This time difference is extreme, as can be seen on Figure 4, and demonstrates how important it is to balance security and performance. They key we will use as a standard for comparison with other results is the 128 bit key, which would take roughly 180 hours to run. It would take more than 11 times the training process of classical FL-GAN to just complete the encryption and decryption with the PC.

Moving on to the CKKS scheme for SHE, we see that there is a huge difference between the two approaches (encrypting every parameter in every tensor and encrypting every tensor as a whole). The first approach took roughly 1347 hours compared to 36 minutes of the second approach. We expected the second approach to be quicker, but this difference in efficiency is shocking. It makes us think of the results we would have gotten if we were able to do the second approach on the Paillier Cryptosystem. This will be expanded in section 8. However, when it comes to comparing approach 1 of CKKS to the PC, we can see that CKKS is more than 7 times less efficient. As CKKS is a lot more complex, and given the fact that we can do both addition and multiplication with it, this does not come as a surprise. There also is a loss in accuracy, which we have unfortunately been unable to measure.

The last results we got were with regards to MPC. As predicted, this provided the least time overhead, only 55 seconds. It was faster than any other differential privacy tool we used in this experiment by far. While there is no reason not to use MPC given such a low performance loss, there does appear to be a slight loss in accuracy (value of 2.2798291e-05 per parameter). This adds up over all the parameters, clients, and epochs, and would most definitely increase our BCELoss. Overall, the security properties are lower for MPC than for Homomorphic Encryption, but the low overhead may make such an option enticing.

Figure 5 shows the difference in extra time overhead for the PC (128 bits), SHE (128 bits) encryption per parameter per tensor, SHE (128 bits) encryption time per tensor, and MPC.

6.2 Missing Results and Improvements

To begin, we are missing any results when it comes to Fully Homomorphic Encryption. While we used the CKKS scheme, which has the bootstrapping option, we never used it. The reason is our federated algorithm did not contain any multiplicative operations. Moreover, we only had 3 clients, making bootstrapping unnecessary. It would also have required a lot more time, making it difficult to fit within our time constraints.

Another improvement that could have been made was using the DelftBlue cluster [10]. The miscalculation here was missing out on the fact that every MPI node runs on one CPU, thereby no mater how many CPUs total we had there was no performance boost. Had we used a different implementation,

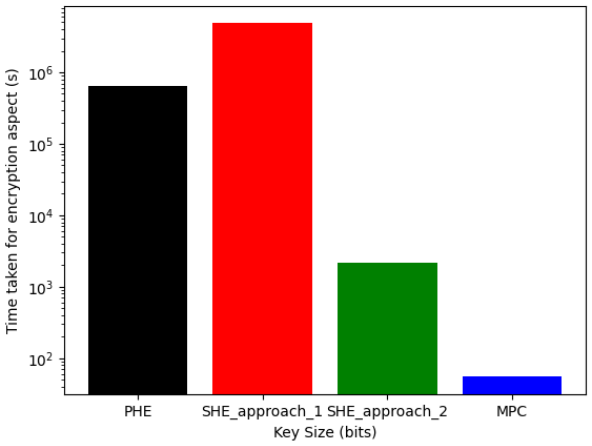

Figure 5: Total time lost between PHE, SHE, and MPC

we could have been able to run some of the implementations rather than predicting their results.

7 Responsible Research

While initially studying and reading through research papers and articles, an effort was made to ensure the papers were written by different authors performing their research from different parts of the world. That was done to ensure our starting point was as unbiased as possible, and our knowledge diverse to the greatest extent. All of the resources used are publicly available, and can be accessed by anyone. Moreover, we have attempted to explore technologies developed by industry standard companies, such as Facebook/Meta and Google. We believed studying the work of such reputable companies would decrease the probability of any bias being included in the research. They have the resources and responsibility to ensure their work is at the highest possible level.

Furthermore, there was no personal bias involved when implementing the FL-GAN with HE. Our sole objective was to study the effect of HE on the performance of training such ML models. We used the most famous tool we could find, which were all open source. We also went over a considerable amount of repositories that we attempted to use unsuccessfully, which can be recreated by anyone. We also thought that including it in the paper would save any future research valuable time when trying to solve a similar problem.

Lastly, while our study focuses on the impact of the performance loss of using HE on FL-GANs, the whole idea of using HE is to increase the privacy of the end users. That is a hot topic in today's world, and motivates us to be extra delicate and careful to ensure the most accurate results and implementations. While our implementation is not 100% real world applicable and is more research based, we hope that it may be helpful with creating an implementation that is.

8 Future Work

This section will go over what we believe should be then next when it comes to our research topic.

The first thing we believe should be done is trying out the effect of HE on FL-GANs using different implementations. There do exist a lot of other forms of repositories that allow for us to make an FL model, such as Fate [11], and they all may have their own tools that are optimized for implementing HE. It would be interesting to compare amongst the different forms of implementation.

Also, given that most HE tools (such as Microsoft SEAL) are written in c and c++, which are both programming languages that have better optimizations than python, one could implement the whole project in those languages. On the other hand, most ML models are implemented with Python, so choosing different languages would mean less online support and less flexibility to alter the models that do exist.

When it comes to our implementation using PyTorch, it would be ideal to test it with PySyft when the documentation is updated. That would give us the options of encrypting tensors as a whole using the Paillier Cryptosystem, which would give individuals and corporations more knowledge on how big of an impact HE would actually have on the models.

Moreover, as mentioned throughout this paper our FL was implemented through MPI. An industry standard FL-GAN would be implemented in a different way, where the communication costs between the clients and the server would probably take considerable extra time, which we currently do not take into account. Also, with a real implementation we can get the performance metrics of the accuracy of different HE schemes. In addition to that, there is a high likelihood that the system would have considerably more clients than three, which definitely needs to be further looked into and experimented. Using different federated algorithms would also be useful, as they may have a different amount of arithmetic operations on the ciphertexts. This could affect a leveled CKKS scheme, especially when combined with more clients.

Finally, it would be crucial to also experiment on FHE. As this is the strongest form of HE, finding out how much slower it is than SHE and PHE would be significant. As mentioned in the previous paragraph, this should also be done for different federated algorithms.

With all of the above mentioned sections, it should be noted that more time should be spent researching the accuracy loss of training the model, as the results in our experiment were more focused on performance.

9 Conclusion

When it comes to the training of Machine Learning models, nothing trumps the importance of privacy and data confidentiality of individuals. While Federated Learning is a significant improvement as only parameters of a model leave the model training location, it is still not ideal as bad actors are in certain situations able to reverse engineer the parameters to obtain the original data. One of the solutions to this issue is using Homomorphic Encryption, an encryption technique allowing us to perform computations on ciphertexts. This way, we only communicate encrypted parameters, which ensures that no unauthorized party has any access to the initial data used to train the ML model.

The objective of this paper is to find the performance loss

of training a Federated Learning Generative Adversarial Network using Homomorphic Encryption. From the three types that exist (Partial, Somewhat, and Fully HE), we were able to predict the performance loss of Partial and Somewhat HE (for our implementation). We have used the Paillier Cryptosystem for PHE and the Cheon-Kim-Kim-Song for SHE. All of our results are for a FL structure consisting of three clients, but where applicable we have also added a formula that can be used to estimate the performance loss for a n-client system. Moreover, we have also predicted the performance loss of Multi Party Computation, as it has homomorphic properties. Our results support the fact that as our encryption system gets stronger, the performance loss is higher, making the decision of balancing security and performance a difficult but nevertheless vital issue for the developers.

As we implemented and worked on our research topic, we have come across many different FL implementation and HE libraries that are optimized for those implementations. That opens the possibility for a future research on those industry standard tools, and comparing the results amongst themselves. It will be interesting to see the effect on efficiency as the homomorphic encryption techniques gets more optimized.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[2] The TensorFlow Federated Authors. TensorFlow Federated, 12 2018.

[3] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption. *CoRR*, abs/2104.03152, 2021.

[4] Zhengjun Cao and Lihua Liu. The paillier's cryptosystem and some variants revisited. *CoRR*, abs/1511.05787, 2015.

[5] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[6] Anamaria Costache, Benjamin R. Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. On the precision loss in approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2022/162, 2022. https://eprint.iacr.org/2022/162.

[7] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, Jan 2018.

[8] Lisandro Dalcin and Yao-Lung L. Fang. mpi4py: Status update after 12 years of development. *Computing in Science Engineering*, 23(4):47–54, 2021.

[9] data61. python-paillier, 2022.

[10] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[11] FederatedAI. Fate, 2022.

[12] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[14] Inferati. Introduction to the bfv encryption scheme, Feb 2021.

[15] Jichu Jiang, Burak Kantarci, Sema Oktug, and Tolga Soyata. Federated learning in smart city sensing: Challenges and opportunities. *Sensors*, 20:6230, 10 2020.

[16] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2019.

[17] Surya Teja Karri. Dcgan-cifar10-pytorch, 2019.

[18] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *CoRR*, abs/2109.00984, 2021.

[19] Xiling Li, Rafael Dowsley, and Martine De Cock. Privacy-preserving feature selection with secure multiparty computation. *CoRR*, abs/2102.03517, 2021.

[20] Zhuohang Li, Jiaxin Zhang, Luyang Liu, and Jian Liu. Auditing privacy defenses in federated learning via generative gradient leakage, 2022.

[21] Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. Privacy-preserving federated learning based on multi-key homomorphic encryption, 2021.

[22] Jaehyoung Park and Hyuk Lim. Privacy-preserving federated learning using homomorphic encryption. *Applied Sciences*, 12(2), 2022.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[24] Mohammad Rasouli, Tao Sun, and Ram Rajagopal. Fedgan: Federated generative adversarial networks for distributed data, 2020.

[25] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized federated averaging. *CoRR*, abs/2104.11375, 2021.

[26] Febrianti Wibawa, Ferhat Ozgur Catak, Salih Sarp, Murat Kuzlu, and Umit Cali. Homomorphic encryption and federated learning based privacy-preserving cnn training: Covid-19 detection use-case, 2022.

[27] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, Theo Ryffel, Zarreen Naowal Reza, and G. Kaissis. Pysyft: A library for easy federated learning. 2021.