Performance Impact Analysis of Homomorphic Encryption: A Case Study Using Linear Regression as an Example

Thomas Prantl[1], Simon Engel[1], Lukas Horn[1], Dennis Kaiser[1], Lukas Iffländer[1], André Bauer[1], Christian Krupitzer[2], and Samuel Kounev[1]

[1] University of Würzburg, Germany firstname.lastname@uni-wuerzburg.de
[2] University of Hohenheim christian.krupitzer@uni-hohenheim.de

Abstract. In recent years, the trend has increasingly been to store and process data in the cloud. However, this is based on the premise that cloud providers treat the data in a trustworthy manner. One way of using the data in the cloud without the provider having access to it is homomorphic encryption. However, since this encryption has only recently become practicable, analysis of its for practical applications is still in its infancy. Therefore, we investigate the performance of homomorphic encryption using a real-world application, namely linear regression. Our main finding is that although the homomorphic computation of linear regression is in the range of minutes and thus slower than in the non-homomorphic case, linear regression can be computed homomorphic and is therefore suitable for use cases where data security is the top priority.

Keywords: Homomorphic Encryption · Performance Analysis

1 Introduction

In our digital world, the collection and processing of data is of paramount importance to businesses. To provide a flexible and scalable infrastructure for the data, the trend is to utilize cloud computing. In fact, Marc Hurd (former co-CEO of Oracle Corporation) estimates that 80% of enterprise data centers will be moving to the cloud by 2025 [17]. However, to use the cloud and its benefits, the data must be given to a third party that must be trusted. After all, there are many fraud scenarios. For instance, the cloud provider could be active in the same area as the user and use the uploaded data himself. To mitigate these risks, one solution could be to use homomorphic encryption. This encryption allows data to be stored and processed in a public cloud while the provider does not have access to it. Technically, due to the homomorphic encryption, the user can then run databases or microservices in the cloud as well as train machine learning models and store the data in the cloud without any concerns. Although the idea of homomorphic encryption was introduced in 1978 [16] and the first implementation of this method [4] was presented in 2009, homomorphic encryption has only recently been made available to developers in the form of corresponding

libraries. As a result, the performance analysis of homomorphic encryption for everyday uses cases is still in its infancy. Consequently, the goal of this paper is to investigate the performance of this encryption in practice, using linear regression as an example. In summary, our evaluation shows that although the use of homomorphic encryption slows down the computation of linear regression, the computations are still possible in an acceptable time and that homomorphic encryption is thus a way to realise data protection. The remainder of this paper is organized as follows: In Section 2, we introduce the concept of homomorphic encription and linear regression. In Section 3, we explain the architectures utilized to assess the performance. Next, we investigate the performance in Section 4. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2 Background

This section introduces the basic terms and concepts of homomorphic encryption, linear regression, and gradient descent.

2.1 Homomorphic Encryption

For the definition of a homomorphic cryptosystem, we first define the term cryptosystem. For this, however, we first need to define the terms plaintext and ciphertext. We call all things that can be encrypted plaintext. The encryption of a plaintext is called a ciphertext. Based on these terms we can define what a cryptosystem is according to [11].

Definition 1. *A cryptosystem is defined as a tuple* $(\Sigma, \mathcal{G}, \mathcal{E}, \mathcal{D})$ *with the following properties:*

- Σ *is a finite, non-empty set. It is called the "alphabet". The following three sets are subsets of* Σ*:* $\mathcal{P}$ *is the "plaintext space",* $\mathcal{K}$ *is the "key space",* $\mathcal{C}$ *is the "ciphertext space"*
- $\mathcal{G}$ *is a probabilistic algorithm that outputs a keypair* (pk, sk) *chosen according to some distribution*
- $\mathcal{E}$ *takes as input a key* k *and a plaintext message* m *and encrypts it to a ciphertext* c
- $\mathcal{D}$ *takes as input a key* k *and a ciphertext* c *and outputs the plaintext* m

Additionally, a cryptosystem must satisfy the following condition, otherwise it is not guaranteed that a ciphertext can be brought back to it's original form, which would make the cryptosystem quite useless:

$$\forall m \in \mathcal{P} : \forall (pk, sk) \in \mathcal{K} : \mathcal{D}(sk, \mathcal{E}(pk, m)) = m$$

In the following, we denote the key as a pair (pk, sk) where pk is the key for encryption and sk the one for decryption. Now that we have introduced the notion of cryptosystem in general, we extend it with respect to homomorphism. Homomorphic encryption aims to do operations like addition, multiplication, exponentiation, etc. on encrypted data. Therefore, we extend the Definition 1 to fulfill these requirements.

Definition 2. *A homomorphic cryptosystem is defined as a tuple* $(\Sigma, \mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{F}, Evaluate)$ *with the following properties [5]:*

- $(\Sigma, \mathcal{G}, \mathcal{E}, \mathcal{D})$ *is a cryptosystem*
- $\mathcal{F}$ *is a set of functions which can be calculated by the cryptosystem*
- *Evaluate is an algorithm that given a key* k*, a function* $f \in \mathcal{F}$ *and a ciphertext* c *calculates a new ciphertext* c' *of the same length*

The last condition is necessary because otherwise it would be possible to just write the desired calculation at the end of the ciphertext and execute it when it is decrypted. We now extend this definition to a fully homomorphic cryptosystem.

Definition 3. *Let* $\mathcal{H} = (\Sigma, \mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{F}, Evaluate)$ *be a homomorphic cryptosystem. It is called "fully homomorphic" if* $\mathcal{F}$ *contains all possible functions.*

2.2 Linear Regression with Gradient Descent

We define the linear regression problem according to [13]. The linear regression problem can be simplified described as follows: Based on a a set of variables x_1, $\dots$, x_n we try to determine the target variable y. In doing so, we assume that we can weight these variables differently and add them up. We also assume that there is a linear relationship between the target and the influencing variables and we also have a corresponding dataset, which depicts concrete influencing variables on the target variable. The problem to be solved now is how to determine the different weights of the influencing variables so that the target variable can be calculated as accurately as possible. More formally defined, we have an influence vector $X = \{x_1, \dots, x_n\}^T$, a weight vector $\Theta = \{\Theta_1, \dots, \Theta_n\}^T$ and a target variable y and have to choose Θ in such a way that the scalar product of Θ and X gives y, as shown in Equation 1a.

$$y = f(X) = \sum_{i=0}^{n} \theta_i x_i \qquad (1a) \qquad J(\Theta) = \frac{1}{2} \sum_{i=0}^{n} (\theta_i x_i - y_i)^2 \qquad (1b)$$

One method to determine Θ is the gradient descent method [13], which requires the existence of a derivable error function, also called cost function. For this purpose, we use the function in Equation 1b analogous to [13]. The procedure of the gradient descent method can be summarized as follows: An initial random assignment is chosen for Θ. Then, the target variable is predicted for each data point of the dataset and Θ is updated based on the deviation. The update of a concrete $\Theta_j \in \Theta$ is thereby done by according to Equation 2. Using our cost function Equation 2 can be simplified to Equation 3. In practice, the dataset is not only iterated once but several times to update Θ. In the following, we also refer to an iteration as a learning epoch.

$$\Theta_j := \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta) \qquad (2)$$

$$\frac{\partial}{\partial \Theta_j} J(\Theta) = \Theta_j - (\theta_j * x_j - y_j) * x_j \qquad (3)$$

3 Evaluation Architectures for Homomorphic Linear Regression

In this section we present our evaluation environment. In doing so, we consider two different architectures. As a metric, we consider the time required for the respective calculations. Since exactly the same result parameters were always calculated for the linear regression in the non-homomorphic and the homomorphic case, we do not go into more detail about the accuracy of both methods.

3.1 Offline Client Architecture

The functionality of the offline architecture is illustrated in Figure 1(a) and shows the two parties involved, the client and the server. The client wants the linear regression to be calculated on its dataset, but does not want to perform these calculations itself. Instead, the client wants the server to perform these calculations for it. However, the client wants the server to perform the calculations, but not to see the dataset or know the results of the calculations. For this reason the client encrypts its data homomorph and sends it to the server. The server can perform the required calculations on the homomorphic encrypted data and send the result back to the client. The client then only has to decrypt the result.

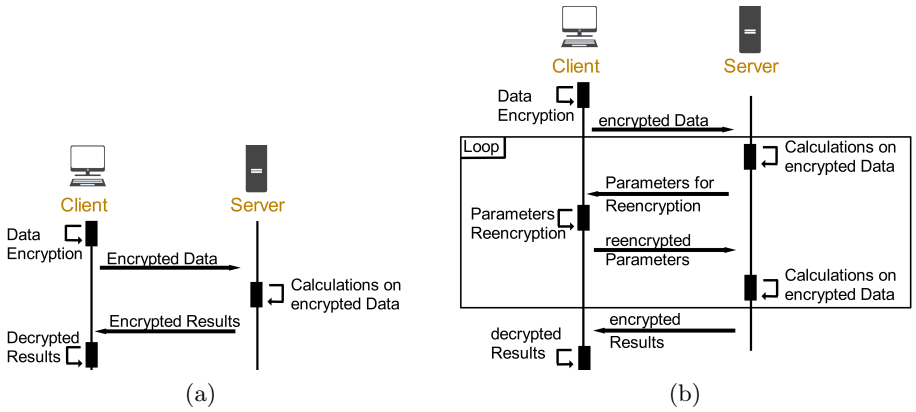


Fig. 1: Illustration of the (a) offline and (b) online architecture.

As metrics for the offline architecture, we use the time $\bar{t}_{prep}$ it takes to encrypt the dataset homomorphic, the time $\bar{t}_{c_e}$ it takes to calculate the linear regression, and the time $\bar{t}_d$ it takes to decrypt the results. We calculate $\bar{t}_{prep}$, $\bar{t}_{c_e}$, and $\bar{t}_d$ respectively by homomorphic encryption the original dataset n times in a row, or performing the homomorphic linear regression calculation n times in a row, or decrypting the results n times in a row and averaging over the required times. As an example of these calculations, Equation 4 gives the calculation of $\bar{t}_{prep}$,

where $t_p^{i,1}$ represents the time at which the initial dataset was encrypted for the i^{th} time and $t_p^{i,0}$ represents the time at which the i^{th} encryption process of the initial dataset began. As a measure of accuracy, we use the standard deviation in each case, which is given for $\bar{t}_p$ in Equation 5 as an example.

$$\bar{t}_{prep} = \frac{1}{n} \sum_{i=0}^{n} t_p^{i,1} - t_p^{i,0} \tag{4}$$

$$\sigma_{prep} = \sqrt{\frac{1}{n-1} * \sum_{i=0}^{n} (t_p^{i,1} - t_p^{i,0} - \bar{t}_{prep})^2} \tag{5}$$

To evaluate the performance impact of homomorphic encryption we consider (1) how do the client computation times behave in the homomorphic case relative to the non-homomorphic case, see Equation 6a and (2) how does the sum of the computation times of the server and client behave in aggregate in the homomorphic case relative to the non-homomorphic case, see Equation 6b. The first case indicates the factor by which the client must wait longer for the results of the linear regression if the client homomorphic outsources the calculations to the server than if the client calculates the linear regression itself. The second case describes the factor by which the total calculation times for the client and server increase if the client homomorphic outsources the calculation of the linear regression to the server instead of calculating it itself. In the Equations 6a and 6b $\bar{t}_{nh}$ stands for the average time the client would need to compute the linear regression on the unencrypted dataset.

$$O_{Client} = \frac{\bar{t}_{prep} + \bar{t}_d}{\bar{t}_{nh}} \tag{6a}$$

$$O_{Total} = \frac{\bar{t}_{prep} + \bar{t}_d + \bar{t}_{c_e}}{\bar{t}_{nh}} \tag{6b}$$

3.2 Online Client Architecture

The online architecture, shown in Figure 1(b), includes the actors client and server. The online architecture is largely identical to the offline architecture, except for one difference. The client assists the server in homomorphic computations. The server can send the client parameters that the client should re-encrypt, which resets the multiplicative depth of these parameters. The re-encryption of parameters can take place as often as needed. As metrics for the online architecture, we use $\bar{t}_{prep}$, $\bar{t}_{c_e}$, and $\bar{t}_d$ and augment them with the time $\bar{t}_{re}$ required by the client to re-encrypt homomorphic encrypted parameters. The computation of $\bar{t}_{re}$ and its accuracy measure is analogous to the Equations 4 and 5. We determine the impact of applying Homomorphic encryption in the case of the online architecture in Equations 7a and 7b largely analogous to the corresponding Equations 6a and 6b of the offline architecture. The only difference is that $\bar{t}_{re}$ is added to the numerator in each case.

$$O_{Client} = \frac{\bar{t}_{prep} + \bar{t}_d + \bar{t}_{re}}{\bar{t}_{Client}} \quad (7a) \qquad O_{Total} = \frac{\bar{t}_{prep} + \bar{t}_d + t_{c_e} + \bar{t}_{re}}{\bar{t}_{Client}} \quad (7b)$$

4 Evaluation

In this section, we present our measurement set up and results.

4.1 Measurement Set Up

For the sake of simplicity, we have realized the client and the server on the same hardware. All the measurements for our two architectures were carried out on our HPE ProLiant DL360 Gen9 server. This server has 8 CPU cores with 2.6 GHz each and 32 GB RAM. We used Ubuntu 20.04 LTS as the operating system. For the implementation of the underlying homomorphic cryptosystem, we used the open source library PALISADE [14]. Since PALISADE only provides the operations addition and multiplication, we had to implement the linear regression ourselves using the gradient descent method. As a dataset on which we want to calculate the linear regression we have used the dataset of the Kaggle competition, in which the value of a house is to be learned [9].

4.2 Offline Architecture Evaluation

For the evaluation of our offline architecture, we first look at the times needed to encrypt datasets of different sizes, followed by the time needed to decrypt the results. Then we analyze at the computation times of the server as well as the overhead of the client and the total overhead.

Dataset Encyption Figure 2a shows that the time needed to encrypt a dataset increases with the size of the dataset, as expected, but also with the number of the trained epochs, which is unexpected at first glance. The reason for this is that we encrypt the data with the minimum level of multiplicative depth required for the respective number of epochs. The multiplicative depth indicates how many multiplications are allowed in each case. When allowing more multiplications, more complex data must be encrypted, which leads to longer encryption times. Additionally, it is noticeable in Figure 2a that encryption times are not given for every combination of dataset size and number of epochs. This is because we have only given the times for the combinations for which we were able to both encrypt the data and perform the subsequent homomorphic calculation of the linear regression with the RAM available on the server. Thus, with a dataset size of 100 elements we were still able to calculate all planned 13 epochs, for 200 elements only 6 epochs, for 300 elements only 4 epochs, for 700 elements only 3

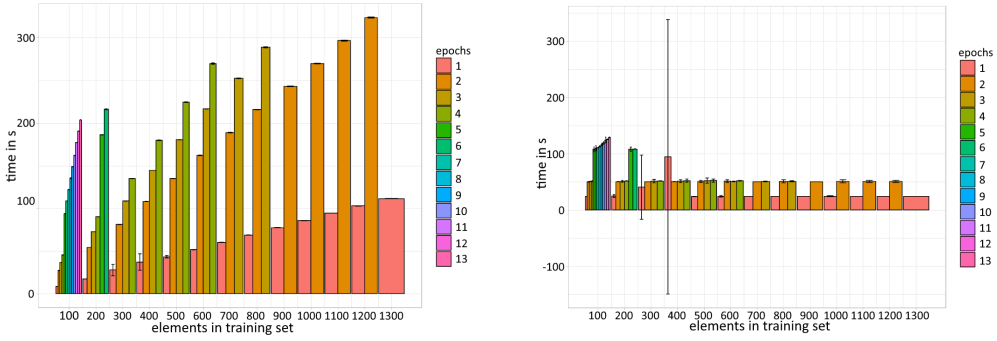
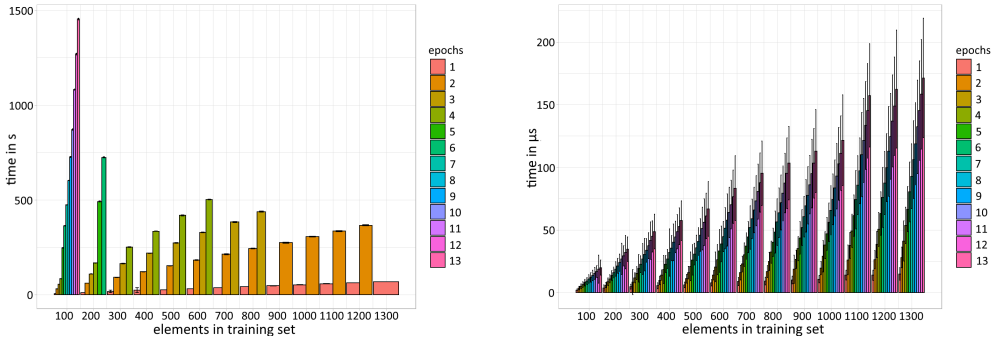


Fig. 2: Times required to (a) encrypt datasets and (b) decrypt results in case of the offline architecture. We used the standard deviation as a measure of accuracy. We have also only given the encryption times of the combinations of number of elements and epochs for which we were able to encrypt the dataset and perform the homomorphic calculations with the given RAM.

epochs, for 900 elements only 2 epochs and for 1300 elements only one epoch. In summary, we draw the following conclusions for the encryption of the dataset: (1) the encryption times depend not only on the dataset size but also on the complexity of the performed operations (here: the required multiplicative depth), (2) homomorphic encryption has large demands for RAM , which is why the linear regression could not be calculated homomorphic for many combinations, and (3) for the feasible combinations, less than 4 minutes were required for the encryption in each case. The 4 minutes alone for preparing the encrypted data before the actual linear regression calculations can begin may seem like a lot at first glance, however, it is important to remember that the encrypted data can also be used for other evaluation and thus represent only a one-time cost.

Results Decryption The times for decrypting the results also depend on the multiplicative depth, but are independent of the dataset size, which can be seen in Figure 2b, in which we have again analogously only given the times for the feasible combinations. The independence of the dataset size is due to the fact that the number of result parameters for linear regression is constant. Analogously to the times for encrypting the dataset in Figure 2a, the times for decrypting the result parameters increase with the number of learning epochs. This is again due to the fact that more complex calculations are required for more learning epochs, which is why the data must be encrypted in a more complex manner to allow a greater multiplicative depth, resulting in longer decryption times. However, since the required decryption times are always less than 200 ms, apart from two outliers, the decryption times are negligible in our opinion.



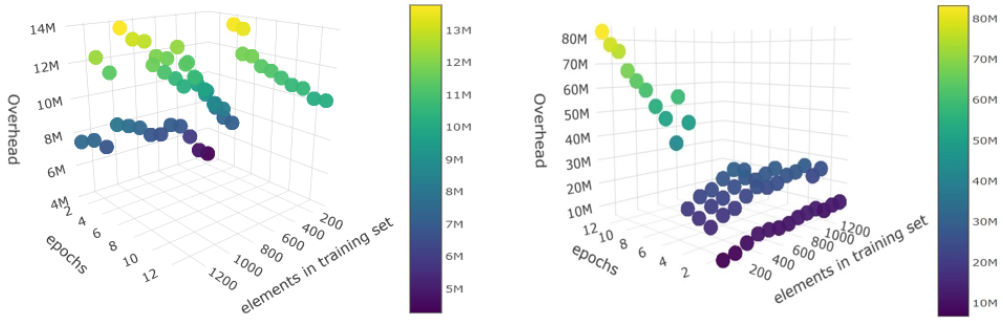
(a) (b)

Fig. 3: Calculation times of the server for linear regression in (a) case of the offline architecture and (b) the non-homomorphic case. The standard deviation in the form of error bars was given as a measure of accuracy

Homomorphic Calculations The times required by the server in the case of the offline architecture for calculating the linear regression homomorphic using the gradient descent method can be seen for different dataset sizes in Figure 3a. In this figure, it is again noticeable that we could not execute the same number of epochs for all dataset sizes with the given RAM of the server. In order to be able to classify the required calculation times in the case of the online architecture, the respective calculation times for the non-homomorphic case are illustrated in Figure 3b. What is immediately apparent from the Figure 3a and 3b are the following two obsevations: (1) in the non-homomorphic case, the server's RAM was sufficient to train 13 epochs each for all dataset sizes and (2) in the non-homomorphic case, always less than 225 μs are required, while in the homomorphic case, the smallest dataset for 13 epochs already requires over 1250 seconds. Thus, the homomorphic calculation of linear regression using the offline architecture are significantly slower than the non-homomorphic calculation.

Time Overhead In order to better quantify the overhead already observed, which the homomorphic calculation of the linear regression entails, we additionally consider O_{Client} and O_{Total} in the Figures 4a and 4b. To do this, we first consider Figure 4a, in which the overhead O_{Client} of the client in the case of the offline architecture is shown depending on the dataset size and performed learning epochs. To reiterate, the overhead O_{Client} defined in Section 2 can be simplified seen as the factor by which the client's computational time is higher due to the use of homomorphic encryption and the corresponding architecture than if the client had computed the linear regression itself non-homomorphic. The time overhead for the client is in all cases at least a factor of more than 4 million but at most a factor of 14 million. The overhead of the client increases the larger the dataset or the number of learning epochs.

The total overhead O_{Total} generated by the use of homomorphic encryption for the client and server is illustrated in Figure 4b. To recap, the overhead O_{Total} defined in Section 2 can be simplified as the factor by which the computation time of the client and the server is higher due to the use of homomorphic encryption and the corresponding architecture than if the client had computed the linear regression itself non-homomorphic. This graph shows that the total overhead in terms of time for the cases considered amounts to a factor between 10 million and 80 million. Similar to the overhead of the client alone, the total overhead increases with increasing number of epochs or dataset size. In summary, it can be said that the overhead in terms of time due to the use of homomorphic encryption in the offline architecture is well over one million.



(a) (b)

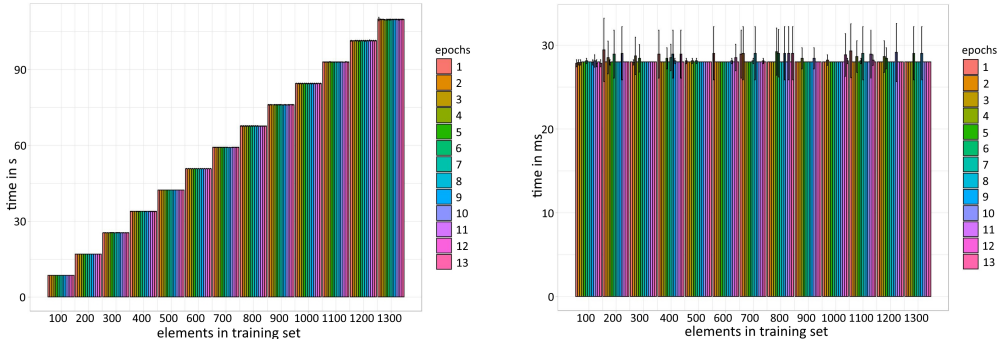
Fig. 4: The total time overhead of (a) the client and (b) the client and server in the case of the offline architecture. Again we have also only given the overhead times for those combinations of number of elements and epochs for which we were able to perform the homomorphic calculations with the given RAM

4.3 Online Architecture Evaluation

For the evaluation of our online architecture, we proceed analogously to the evaluation of our offline architecture and start again with the encryption and decryption times. Afterwards, we compare the times for computation before analyzing the introduced overhead.

Dataset Encryption and Results Decryption Figure 5a and 5b show the encryption and decryption times. Compared to the offline architecture, the encryption times only depend on the size of the dataset and increase linearly with the dataset size. This is because by re-encrypting the data we can reset the multiplicative depth and thus only need the multiplicative depth required for one epoch. This means that we never need more than 2 minutes to encrypt with

the online architecture, whereas we sometimes need almost twice that time with the offline architecture. It is also noticeable that with the online architecture we were able to execute the planned 13 epochs for each dataset size. With less than 2 minutes, the time required for encryption is also negligible in our opinion, since the encrypted data can also be used for further evaluations. The time needed to decrypt the results is illustrated in Figure 5b. Analogous to the offline architecture, the decoding times of the results are independent of the dataset size. However, in the case of the online architecture, they are also independent of the number of learning epochs. This is because the multiplicative depth is now independent of the number of learning epochs and we can therefore use the same depth in each case. Since the result decoding times, also within the scope of the error, are below 40 ms in each case, these are again negligible in our opinion in contrast to the required time for the encryption of the dataset.

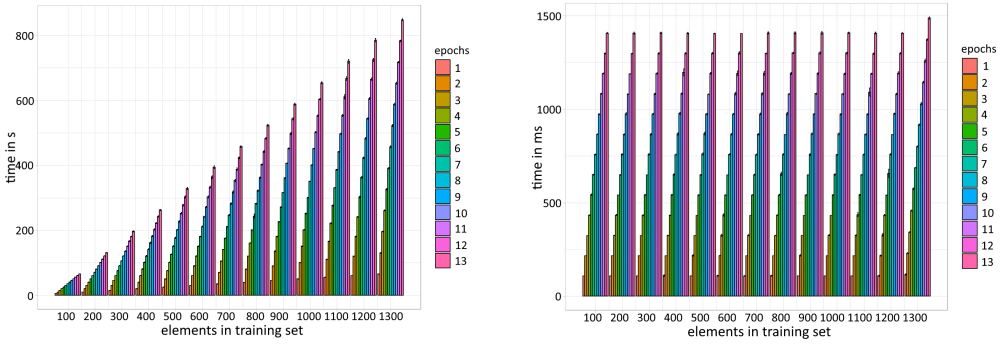


(a) (b)

Fig. 5: Times required to (a) encrypt datasets and (b) decrypt the result of the linear regression in the in case of the online architecture. We used the standard deviation as a measure of accuracy, but it is mostly too small to be visible.

Homomorphic Calculations Analogous to the evaluation of the offline architecture, we next look at the required calculation times of the server in Figure 6a. What is striking in comparison to the calculation times of the server in the case of the offline architecture in Figure 3a is that we were able to train the 13 epochs for all dataset sizes in the case of the online architecture. In addition, the calculation times in the case of the online architecture are significantly shorter than the corresponding times in the offline architecture, but are still significantly greater than the times required in the non-homoropic case, see Figure 3b. This means that the online architecture already provides the server with a performance boost in terms of time in contrast to the offline architecture. However, this boost is only achieved because the client is now ready to re-encrypt intermediate results. This results in additional computation time for the client compared to the offline

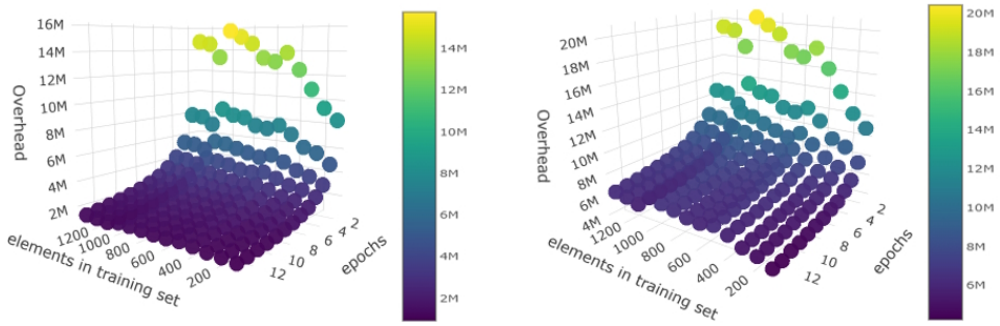
architecture. If we look at Figure 6b, we can see that this additional effort also increases with the size of the dataset and the number of learning epochs, but we can also see, that this effort has always remained below 1.5 seconds in total. Thus, in our opinion, this additional effort for the client is negligible, as the 1.5 seconds is only the cumulative time required, and the client is significantly less burdened for a single re-encryption and is thus still available for other tasks.



(a) (b)

Fig. 6: Calculation times of (a) the server for the linear regression and (b) the client for the re-encryption in the case of the online architecture. The standard deviation is again mostly too small to be visible.

Time Overhead Finally, we look at how the overhead of the client and the total overhead behave in the case of the online architecture. To do this, we first look at the overhead of the client in Figure 7a. Based on this figure, it can be seen that the relationship from the offline architecture, that the overhead increases with increasing learning epochs or elements in the dataset for the client does not apply to the online architecture. In the case of online architecture, the overhead for the client decreases as the number of learning epochs increases. Compared to the offline architecture the actual overhead for the client is also lower in each case, even if it is still in the tens of millions. If we look at the total time overhead for the server and client in Figure 7b, we can draw the same conclusions for the total overhead as for the client overhead. In summary, the online architecture generates less overhead than the offline architecture in that the client not only encrypts the initial dataset and decrypts the results, but also assists the server with the calculations. However, even with the online architecture, the overhead in terms of time is factors of over one million. The greatest advantage of the online architecture is therefore that it allows up to 13 learning epochs to be calculated for larger datasets. More than 13 epochs would have been theoretically possible, but for time reasons we have limited ourselves to a maximum of 13 epochs.



(a) (b)

Fig. 7: The time overhead of the (a) client and (b) the client and server in the case of the online architecture for the homomorphic calculation of the linear regression.

4.4 Discussion

Our results show that the encryption is feasible in a reasonable amount of time. However, the required time depends not only on the dataset size, but also the operation type. The decryption process requires a negligible amount of time. Furthermore, the computation of the linear regression is significantly slower than the non-homomorphic calculation. In the following, we discuss these findings.

Homomorphic encryption, besides making linear regression slower, is a possibility how to realize data protection. For example, it is extremely difficult to use machine learning in the cloud for research with medical data, because one first need an ethics council and have to comply with all the regulations due to the sensitivity of the data. By means of homomorphic encryption one could share such encrypted data and could also organize, e.g., AI competitions as it is often usual in the AI world. Remarkably, the development of homomorphic libraries is at its beginning. There are already first approaches to adapt homomorphic encryption on GPUs, where already speed ups of the factor 1000 are possible [7]. Optimizing the libraries even further and run on GPU rather than CPU then there is still room for improvements. One possible improvement could be bootstrapping, i.e. that one resets the multiplicative depth of an encrypted vector directly and does not have to decrypt and re-encrypt the vector for this. We have seen that it does not matter how many elements a vector consists of. Thus homomorphic encryption is well suited for applications with parallel computations. Therefore, it would make sense to try homomorphic encryption on neural networks, e.g., by starting with feed forward networks, where one could represent the transition from one layer to the next as a single vector multiplication.

4.5 Threats to Validity

Regarding possible threats to the validity of our statements, we encountered the following points. First, the calculation of overhead depends on the hardware used.

For example, in our example, the total time overhead O_{total} could be reduced by using more powerful hardware for the server, which would allow the server to compute the homomorphic linear regression calculations faster. Thus, the term $\bar{t}_{c_e}$ in the Equations 6b or 7b would be smaller, making O_{total} smaller overall. We are aware of this fact, however, see this as common practice to report the times and, for reproducibility reasons, we also report the used hardware infrastructure. Further, as we rely an a hardware setup with seems to be realistic for practical usage, we think that we limit this threats by that. The second possible danger for the validity of our statements lies in the used library PALLISADE. During our measurements, we noticed that the library contains minor memory leaks. These memory leaks were not due to us, since they occur even in the PALISADE standard examples. To make sure that these memory leaks do not have any influence on the performance, we have carried out corresponding measurements and could not detect any influence on the calculation times. In addition, to make sure that the measurements are not falsified by the memory leaks, we controlled the measurements via an external script, which manually releases the RAM and resolves the memory leaks.

5 Related Work

In this section, we review related work and highlight the novelty of our contribution. The authors of the papers [8] and [15] analyzed homomorphic linear regression as well. However, they used the Paillier cryptosystem, which is not fully homomorphic. The Paillier cryptosystem allows for the product of two ciphertexts to be decrypted as the sum of the corresponding plaintexts and for a ciphertext raised to the power of a plaintext to be decrypted as the product of the two plaintexts. Thus, to compute the multiplication for linear regression, the data must be in plaintext, which is not the case in our fully homomorphic approach. Using a fully homomorphic cryptosystem allows us to compute the multiplications and additions needed for linear regression directly on the encrypted data. By using a cryptosystem that is not fully homomorphic, the authors also do not have the associated problem of increasing multiplicative depth and the performance issues that come with it. We have analyzed these performance problems using two different architectures. We also consider a different application case. We consider the case where the owner of the data can store it homomorphic encrypted in a cloud and use the computational resources there to perform arbitrary analysis, such as linear regression, on that data. In contrast, the authors in [8] pursue the goal that if the dataset is distributed across multiple parties, it is still possible for the individual parties to compute the linear regression together without having to make their data available to any of the other parties. The authors in [15] have a similar goal.

The authors of [12] have also dealt with homomorphic regression, but with logistic regression. As a cryptosystem, they use the system of Cheon [3], which requires only homomorphic encoded data for the computation of multiplication and addition. The authors circumvent the problem of multiplicative depth on the

one hand by rescaling the encrypted values and on the other hand that certain hyperparameters have to be determined beforehand on the unencrypted data. We differ from this approach in that we do not perform precomputations on unencrypted data and analyze the impact of multiplicative depth on performance. In terms of metrics, we additionally consider the overhead incurred by the use of homomorphic encryption and, unlike [12], achieve the same accuracy with our homomorphic encryptions as in the unencrypted case.

In [2], the authors also want to compute the linear regression homomorphic, using Microsoft's SEAL library, which allows multiplication and addition on homomorphic encoded data. The linear regression was computed on different datasets whose size is unknown. Therefore, the performance values given are difficult to classify with regard to the times required for encrypting and decrypting or for calculating the linear regression. In addition, the calculation of the inverse of a matrix, which is necessary for how they calculate the linear regression, is not encrypted. We differ from this approach, among other things, in that all calculations are done homomorphic, we specify the used dataset, and that we analyze the overhead that arises from the use of homomorphic encryption.

The paper [1] is an extension of the work [6]Therefore we focus only on the paper [1]. The authors of [1] make use of the two-server model from [10] and assume that the data owner can outsource the computation of linear regression to two non-colluding servers S1 and S2. In doing so, server S1 combines the data homomorphic encrypted by the data owner and masked them. The data thus encrypted and masked can be decrypted by server S2 to compute the linear regression on the unencrypted but masked data. This approach poses the risk that the servers could collaborate and thus gain access to the data. This is also the main difference with our approach, which performs all computations only on the homomorphic encrypted data. As a result, the data owner retains full control over his data, no matter how many servers of the cloud cooperate.

6 Conclusion

One way of using the data in the cloud without the provider having access to it is homomorphic encryption. However, since this encryption has only been practicable recently, there is only little work that investigates its performance. Therefore, we presented two approaches to realize homomorphic encryption. The first approach is that the client sends its data homomorphic encrypted to the cloud, which is to perform the corresponding calculations. The client does not support the cloud in the calculations and only decrypts the result. The second approach is largely the same as the first approach, except that the client now supports the cloud in the calculations in such a way that it decrypts and re-encrypts individual parameters in order to reset the multiplicative depth of these parameters. We compared those approaches in an linear regression application and analyze the times required for encryption, decryption and computation as well as the introduced overhead. The results show that the encryption times depend not only on the dataset size, but also the operation type; however, in our

scenario, encryption is feasible in a reasonable amount of time. The decryption process requires a negligible amount of time. Furthermore, the computation of the linear regression is significantly slower than the non-homomorphic calculation up to a factor of 80 million.

References

1. Akavia, A., et al.: Linear-regression on packed encrypted data in the two-server model. In: Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography (2019)
2. Chen, B., et al.: Implementing linear regression with homomorphic encryption. Procedia Computer Science **202**, 324–329 (2022), international Conference on Identification, Information and Knowledge in the internet of Things, 2021
3. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 409–437. Springer International Publishing, Cham (2017)
4. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing (2009)
5. Gentry, C.: Computing arbitrary functions of encrypted data. Communications of the ACM **53**(3) (Mar 2010). https://doi.org/10.1145/1666420.1666444
6. Giacomelli, I., et al.: Privacy-preserving ridge regression with only linearly-homomorphic encryption. In: Applied Cryptography and Network Security. pp. 243–261. Springer International Publishing, Cham (2018)
7. Goey, J.Z., et al.: Accelerating number theoretic transform in gpu platform for fully homomorphic encryption. The Journal of Supercomputing **77**(2) (2021)
8. Hall, R., et al.: Secure multiple linear regression based on homomorphic encryption. Journal of Official Statistics **27**(4), 669 (2011)
9. Kaggle: House Prices - Advanced Regression Techniques, online available under https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data, Accessed on 29.10.2022
10. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive (2011)
11. Katz, J., et al.: Introduction to Modern Cryptography. Chapman Hall, CRC Cryptography and Network Security, CRC Press, Boca Raton ; London ; New York, second edition edn. (2015)
12. Kim, M., et al.: Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR medical informatics **6**(2), e8805 (2018)
13. Ng, A.: Cs229 lecture notes. CS229 Lecture notes **1**(1), 1–3 (2000), accessed online 25.09.2022 (Link: https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf)
14. PALISADE Project: PALISADE HOMOMORPHIC ENCRYPTION SOFTWARE LIBRARY, online available under https://palisade-crypto.org/, Accessed on 17.01.2023
15. Qiu, G., et al.: Privacy-preserving linear regression on distributed data by homomorphic encryption and data masking. IEEE Access **8** (2020)
16. Rivest, R.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11) (1978)
17. Steve, R., Angus, L.: Oracle ceo hurd says 80% of corporate data centers gone by 2025, online available under https://www.wsj.com/articles/BL-CIOB-11316, Accessed online 25.09.2022