Homomorphic Encryption and Machine Learning - Final Report

Aditi Dandekar
aadandek@ncsu.edu
North Carolina State University
Raleigh, North Carolina

Shreya Parikh
scparik2@ncsu.edu
North Carolina State University
Raleigh, North Carolina

Sankalp Gaharwar
ssgaharw@ncsu.edu
North Carolina State University
Raleigh, North Carolina

1 ABSTRACT

Machine Learning techniques are being extensively used in multiple domains to analyse the associated data and present insights to optimize performance. However, application of Machine Learning models on sensitive data-sets presents extensive risks to user-privacy owing to the high probability of compromising sensitive user data, when it is provided to the Machine Learning pipeline. To counter this threat, we propose the use of a customized Homomorphic-Encryption scheme that allows for adequate encryption of the user-data through the use of a scheme that employs Public and Private Keys. We analyze and implement a basic Fully Homomorphic Encryption model. We also apply the Homomorphic Encryption scheme on a Linear Regression model for sensitive Medical data. Thus, we observe that the encryption efficiently protects the sanctity of the input test data and corresponding result data for a Regression model. It provides immunity from both Model-Inversion and Membership-Inference attacks, for the ML model and the associated sensitive user-data.

2 PROBLEM MOTIVATION

Successful application of any Machine Learning model on sensitive data represents a major research challenge owing to the various regulatory constraints, which are required to be respected within data-sets associated with domains such as Finance and Healthcare. For such data-sets, the user data often contains sensitive data attributes, which entails enhanced focus on ensuring the privacy of the data. Thereby, when we use this data for training a Machine Learning model such as Linear Regression,the data-dependence of the model on the sensitive user data makes it unsuitable to build Machine Learning pipelines without compromising user privacy.

These concerns are valid for any Machine Learning technique/model that interacts with sensitive data. Additionally, as Machine Learning models are being used on a widespread basis, they are becoming increasingly vulnerable from the standpoint of Privacy. Privacy-oriented attacks on Machine Learning models such as Model Inversion attacks and Membership Inference attacks represent credible threats to Machine Learning frameworks that seek to respect all aspects of user-privacy.

Machine Learning models, such as Linear Regression, rely heavily on data to perform training of the model and subsequently, to drive inference through the model. Therefore, if the data being fed to the Machine Learning model is of sensitive nature (For instance, financial information of users such as Credit Scores), then we cannot use the Machine Learning models efficiently since the user privacy would be compromised. A major concern is that trained models may leak sensitive data, when challenged by practices such as Model Inversion and Membership Inference. For frameworks using Machine Learning-as-a-service, the privacy of the Machine Learning model itself can be endangered through techniques that compromise privacy.

Additionally, if we seek to prevent access to sensitive data attributes for the Machine Learning model with view towards respecting the privacy of training data, we end up endangering the efficacy of the Machine Learning model. Thus, we encounter a trade-off between Data Privacy of the users and the efficiency of the data that we can provide to the ML models.

To prevent attacks such as Model-Inversion on the data associated with Machine Learning models, we must seek to encrypt not only the input training data, but also the output that we obtain from the Machine

Learning model. Along these lines, utilization of Homomorphic Encryption techniques represents a promising avenue for preserving user privacy when we are using Machine Learning tools on the user data to derive insight.

Homomorphic Encryption is a powerful public encryption methodology, that uses iterative Addition and Multiplication operations on the user data, to encrypt it in a manner that allows us to efficiently use any Machine Learning technique over data-sets while preserving user-privacy. through which we can encrypt the user data before sending it to the model, perform computations and analysis on the encrypted data through the Machine Learning model, and finally decode it back on the client-side(i.e. on the local machine of the user) to see the results of the computation on the original user data.

Thus, the sensitive data is encrypted using a public key at the client's machine and sent to the cloud where Machine-Learning models process the encrypted data and derive the requisite insights. Finally, the results of the Machine Learning model are delivered in an encrypted form, where the client uses a private Key to see the results of the analysis, in a form that is consistent with the original form of the sensitive user data. Thus, we are able to maintain privacy as the sensitive data (in original form) is only handled within the secure client network. The computations on the data over the public cloud is only performed when the data is in encrypted form, and hence the possibility of user privacy being compromised is heavily reduced.

We seek to apply multiple variants of Homomorphic Encryption over a data-set containing sensitive user data, to gauge the efficacy of the various models in preserving the performance of a Linear Regression model in a privacy-friendly manner. We analyse multiple Homomorphic schemes and propose a approach through which we can apply Homomorphic Encryption on sensitive data-sets with view towards optimizing user-privacy while ensuring the performance of the Machine Learning model(Accuracy) is not compromized.

3 RELATED WORK

The concept of Homomorphic Encryption was introduced by Rivest, Adleman, and Dertouzos in 1978 [1]. This was the Partially Homomorphic Encryption which allowed addition or multiplication limited number of times. The basic idea was that encryption of plaintext after addition or multiplication is equivalent to that of cipher text after encryption. Somewhat Homomorphic Encryption was later introduced [2] which allowed both multiplication and addition but was limited in terms of number of operations.

A breakthrough in the history of cryptography was the introduction of the first Fully Homomorphic Encyryption Scheme by Craig Gentry in 2009 [3]. This was done by introducing Bootstrapping to Somewhat Homomorphic Encryption. However, this scheme did not lend itself well to practical applications and hence, this has been the focus of extensive research so that we can obtain an encryption scheme that can be applied to practical applications within industries, where privacy is a big concern.

The three most popular schemes which make Fully Homomorphic Encryption practically possible are:

- Brakerski-Gentry-Vaikuntanathan (BGV) [4] scheme : The authors propose that the central concept to their work is the constuction of leveled Fully Homomorphic Encryption Schemes without utilizing Bootstrapping procedures. This technique effectively manages noise level of lattice-based cipher texts.
- Brakerski/Fan-Vercauteren (BFV) [5, 6] scheme: Brakerski introduced a new Tensoring technique for LWE-based Fully Homomorphic Encryption as a result of which noise grows linearly unlike the quadratic growth seen in previous approaches. Fan-Vercauteren introduced two optimized versions of re-linearization which not only give smaller re-linearlization keys but also faster computations. Re-linearization transforms quadratic equations into linear ones.
- Cheon-Kim-Kim-Song (CKKS) [7] scheme: The authors describe the scheme is mainly used for approximate arithmetic operations. The authors state that their decryption structure outputs an approximate value of plaintext with a predetermined precision.

All the three schemes are Lattice-based Cryptographic Schemes whose encryption-security is dependent on the hardness of the Ring Learning with Errors (RLWE)

problem. Lyubashevsky, Peikert and Regev [11] introduced the RLWE problem, which is based on distinguishing polynomially many noisy ring multiplications from uniform distributions. They demonstrate that obtaining the solution for the RLWE problem has parity with obtaining the solution for some worst-case problems in ideal lattices, instead of general lattices

Hence, the RLWE problem[10], as a computational problem, serves as the foundation of new cryptographic algorithms.This problem can be stated in two ways: search way and decision way. In the search version, we aim to find the unknown polynomial from a given list of known polynomial pairs. In the decision version, given a list of polynomial pairs we have to determine whether the polynomials were constructed or designed randomly. This means that all the three approaches are quantum-safe. BGV is more efficient as compared to BFV and CKKS. But it is more difficult to use. They are additively and multiplicatively homomorphic. In case of BGV and BFV, computations can be performed on integers. CKKS can perform computations on floating point numbers as well but with limited precision.

Thus, we seek to explore the possibility of using the Homomorphic Encryption schemes to securely perform mathematical operations on sensitive data sets within a given Machine Learning pipeline. We shall focus extensively on how we can make sure that the algebraic operations that are fundamental for the Machine Learning models can be performed in an efficient manner without providing access to the original sensitive user data points.

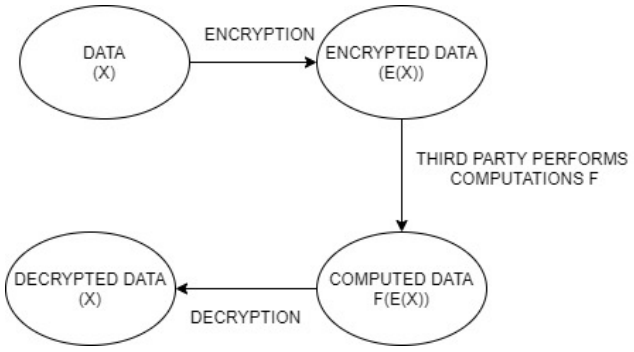

Figure 1: Data Flow Diagram

4 PROPOSED APPROACH

4.1 Rationale for choosing Fully Homomorphic Encryption Scheme

We approached the project by initially studying the Homomorphic Encryption Scheme and its different variants. While researching though the relevant academic literature for solving our problem of preserving privacy while applying Machine Learning techniques on any sensitive data set, we realized the potential of using Homomorphic Encryption Schemes as a promising solution.

Homomorphic Encryption's premise of allowing computation on encrypted data deeply resonated with us as any Machine learning model could then perform its operations with maximum efficacy while we could be confident that our sensitive data is encrypted throughout its exposure to high-risk distributed environments.

Hence, the capability of Homomorphic Encryption to allow computations on encrypted data through the use of tailored algebraic expressions proved to be a game-changer for our problem and immediately made it more favourable for us as opposed to conventional Encryption-Decryption schemes. Once we had settled on using Homomorphic Encryption as our chosen methodology to guard sensitive data, the next logical step for us was to choose a particular variant of Homomorphic Encryption that we would be implementing.

Through our Literature Review, we found three potential Homomorphic Encryption schemes that we could use for our problem: Fully Homomorphic Encryption, Partially Homomorphic Encryption and Somewhat Homomorphic Encryption. We found that the fundamental difference among the three schemes could be attributed to the differing types and frequency of the mathematical operations that we could perform on the encrypted cipher text.

We found that the Partially Homomorphic Encryption scheme allows only one operation, either addition or multiplication, to be performed on encrypted values. This scheme does not put a limit on the frequency with which we can perform mathematical computations on the Cipher text. In contrast, the Somewhat Homomorphic Encryption scheme supports select operation

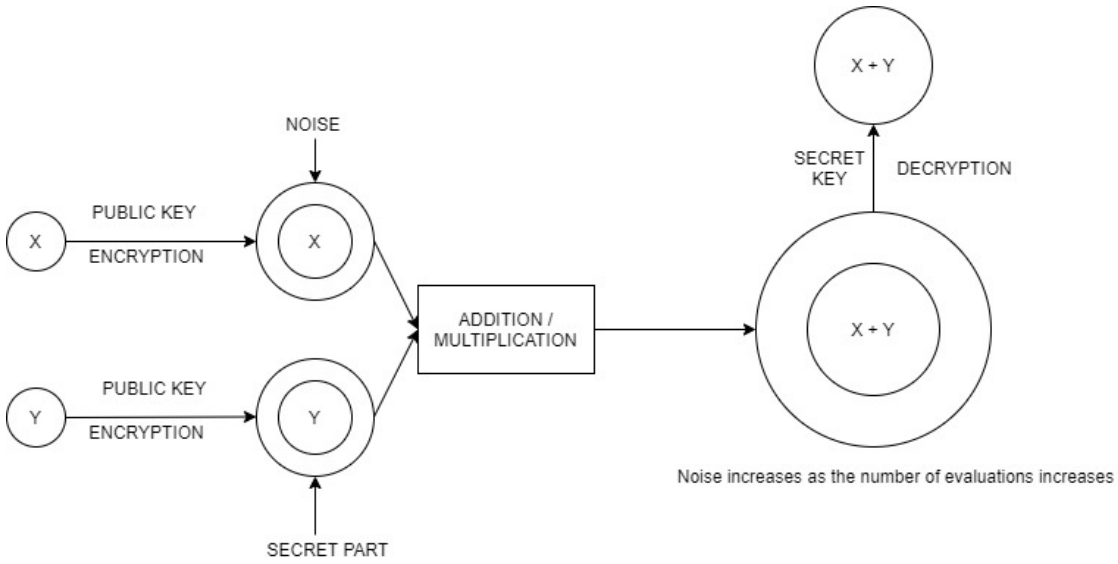

Figure 2: Flow Diagram

(either addition or multiplication) up to a certain complexity. However, the scheme puts a limit on the number of times that we can perform these computations.

Thus, we found the ideal solution to our problem within the Fully Homomorphic Encryption (FHE) scheme that is a refined version of the Somewhat Homomorphic Encryption scheme. FHE is capable of using both addition and multiplication any number of times and hence, is a promising technique for conducting secure multi-party computation in an efficient manner. Additionally, it has the unique capability of being able to handle arbitrary computations on the cipher texts.

Thus,FHE efficiently allows the use of encrypted data for performing useful mathematical operations without access to the original encryption key used for encryption. This lends itself well to ensuring security of applications that run remotely on the cloud. Thus, it is a legitimate solution for our problem of securing sensitive user data before it is fed to the Machine Learning algorithms. Thus, FHE allows the Machine Learning model to pull, search, and manipulate encrypted data without having to allow the ML model access to the original user data.

Additionally, the degree of the security offered by the FHE schemes is beyond reproach since the security of the homomorphic encryption schemes is based on the Ring-Learning With Errors (RLWE) problem : a hard mathematical problem related to high-dimensional lattices. Due to the secure and efficient manner of data-access that FHE provides, it is widely used for applications such as submitting private queries to search engines, providing data access to encrypted files on remote file servers and more generally, in any application that requires secure multi-party computation. Thus, we can efficiently use FHE for providing secure access to sensitive user-data within any Machine Learning model.

Fully Homomorphic Encryption allows us to perform calculations on encrypted data without the need for decrypting it first. A ring homomorphism between two rings satisfies the two properties given below:

- h(x + y) = h(x) + h(y)
- h(x * y) = h(x) * h(y)

Consider we have an encryption homomorphism 'e' and a decryption homomorphism 'd', such that d(e(x)) = x. There is a function f which is a composition of addition and multiplication then,

- Data x is encrypted by user using encryption e to obtain e(x) which is sent to a third party
- The third party performs computations f on e(x). As e is homomorphic encryption, we have e(f(x)) = f(e(x)).
- The data is sent back to the user who decrypts the data such that d(e(f(x)) = f(x). Thus, the data

is not seen by the third party which resolves the purpose of homomorphic encryption.

4.2 Preliminary implementation of Homomorphic Encryption Scheme in Python

In order to better understand Homomorphic Encryption, we implement one of its schemes in Python. Our implementation has basic functionalities of key-generation, encryption, decryption and computations (in the form of addition and multiplication).
The code can be found here : Code
Aim: Basic Implementation of Homomorphic Encryption Schemes using basic mathematical concepts.

The entire approach is divided into the following:

- **Key Generation**: Generation of Public and Secret Keys.
  Secret Key Generation(sk): Generate a random key using uniform distribution over R2 which implies that sk will be polynomial with coefficient 0 or 1.
  Public Key Generation(pk): A polynomial 'a' uniformly over Rq and a small error polynomial 'e' from a discrete normal distribution.

  $$pk = ([(a * sk + e)]_q, a)$$

  Various probability distributions are used to generate keys (Binary polynomial, Uniform Polynomial, Normal Polynomial). The public key is used for encryption while secret key is used for decryption.
- **Encryption**: Encryption of polynomials in the ring R_t .

  $R_t = Z_t/< x^n + 1 >$,where t is the plaintext modulus.

  Encode the integer into the plaintext(pt) domain R_t i.e. constant polynomial m(x) = pt
  Inputs: Public key $pk \epsilon R_q X R_q$; $m \epsilon R_t$
  Output: Ciphertext $ct \epsilon R_q X R_q$
  Mathematical Equation:

  $$ct0 = [pk_0 * \mu + e_1 + \delta * m]_q$$
  $$\text{ct1} = [\text{pk}_1 * \mu + e_2]_q$$

  μ : normally distributed over R_2
  e_1, e_2 : normally distributed over R_q
  δ : integer division of q over t
- **Decryption**: Secret key is used for decryption.

$$\text{Decrypted data} = [\lfloor \tfrac{1}{\delta} \cdot [\text{ct}_0 + \text{ct}_1 \cdot \text{sk}]_q \rceil]_t$$

When we decrypt the data, the correct value is obtained if the rounding to the nearest integer is not affected by the error terms. This means that the error term needs to be bounded by ½. It is observed that the error term must be bounded by q/2t for a correct decryption. Therefore, the value of q and t directly impact the correctness of decryption.
The error terms are generated using probability distributions so the distribution must be chosen carefully. We could choose the parameters easily if we just had to decrypt on the encrypted data but in this case, we also must perform operations on the encrypted data. The computations enlarge the error term. Therefore, the parameters will be chosen according to the number of computations involved.

- **Computation**: Addition/ Multiplication of another plain text.
  Addition: Adding the cipher text(ct) with a plain text(m2). For this operation we need to scale the plain text by δ and add it by ct_0.

  $$\text{addPlain(ct,m2)}=([ct_0 + \delta * m_2]_q, ct_1)$$

  Multiplication: Multiplication the ciphertext(ct) with a plaintext(m2).

  $$\text{mulPlain(ct,m2)}= ([ct_0 * m_2]_q, [ct_1 * m_2]_q)$$

  Even if we add a big polynomial to m1 x m2, the decryption will clearly fail. To get correct decryption we need to multiply ct1 with m2. Comparing to addition, we can observe that the error term got scaled up by plaintext m2.

4.3 Application of Fully Homomorphic Encryption for preserving privacy in a Machine Learning model(Linear Regression)

The code can be found here : Code
Homomorphic Encryption schemes are mainly popular with sensitive datasets. For example, they can be used for Finance or Medical dataset. As a part of this project, we have applied Homomorphic Encryption for

the problem of Linear Regression to the popular Diabetes dataset[8] which belongs to the field of medicine. Diabetes dataset consists of ten baseline variables - age, sex, body mass index, average blood pressure and six blood serum measurements which were obtained for each of the n=442 diabetes patients. The response variable here is quantitative measure of disease progression one year after the baseline. We have used sklearn.dataset package to load the diabetes dataset. The input variables are mean-centred and have sum of square value equal to 1.

Linear Regression is a linear approach to model between dependent variable and one or more independent variables. If there is only one exploratory variable, then it is called simple linear regression. As we are trying to predict disease progression after an year, the current problem at hand is a type of Linear Regression. We split the entire dataset into training and testing sets. As the dataset is very small in size, we consider 30 samples as test samples and the remaining as training samples. We have used sklearn's LinearRegression[9] model for fitting the data. Sklearn's LinearRegression fits a linear model with coefficients to minimize residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation[9].

Body Mass Index (BMI) is widely define obesity and to predict its complications like diabetes and hypertension. Hence, we used Body Mass Index (BMI) to predict diabetes disease progression after an year. This also simplifies the problem to a type of Simple Linear Regression. We fit the model and identify the coefficients and intercept. This helps us create the linear regression equation for prediction.

$$diabetesProgression = (938.2378 * bmi) + 152.9188$$

We have Pyfhel[12] which stands for Python for Homomorphic Encryption Libraries. The library allows Addition, Subtraction, Multiplication and Scalar Product over encrypted vectors or scalars of integers and binaries. The Body Mass Index (BMI) for which we want to calculate diabetes progression is considered to be sensitive data. Thus, it is encrypted using the Pyfhel library. The encrypted data is sent to the model for prediction. The model substitutes encrypted data in the diabetes progression linear equation and sends back data in encrypted form itself. Then the user can decrypt the obtained disease progression.

Thus, in this case privacy of sensitive medical data corresponding to the user like Body Mass Index (BMI) and the obtained diabetes disease progression is preserved using Homomorphic Encryption.

4.3.1 Opensource Homomorphic Encryption Libraries. For implementing our proposed approach, we utilized the Pyfhel library which acts as a wrapper for the various functionalities entailed by the popular open-source Homomorphic Encryption libraries such as SEAL, PALISADE, and HELib. HELib is the most popular open-source HE library that implements the BGV scheme [4] with Smart-Vercauteren ciphertext packing techniques and some new optimizations. HELib is designed through low-level programming procdures (Mainly through C++) and hence, is often referred to as the "Assembly Language for HE". Most of the popular HE libraries build upon the functionalities offered by HELib and offer better implementation and parameter selection capabilities as opposed to the relatively complex HELib implementations.

Along these lines, our implementation through the python-based Pyfhel library heavily relies on Microsoft's Simple Encrypted Arithmetic Library (SEAL), a library derived from the original HELib framework. SEAL is a well documented library, that was built for handling sensitive data with ease-of-use in mind. The library does not have any external dependencies and it includes features such as automatic parameter selection and noise estimator tools, which facilitate its use even with limited background in cryptography. The Pyfhel library also derives features from PALISADE, a HE library for general lattice cryptography that incorporates features such as GPU acceleration for performance optimization.

5 EVALUATION

We evaluate both our implementations namely Preliminary Homomorphic Encryption scheme and application of Homomorphic Encryption in a Machine Learning model separately.

5.1 Evaluation: Preliminary implementation of Homomorphic Encryption Scheme

We test our approach using simple arithmetic. The terminal output of the algorithm is shown in Figure 2. We

take two numbers as plain-text - 73 and 20 respectively. The numbers are encrypted by the algorithm. The output of encryption is printed. Next, we perform simple addition and multiplication on the cipher texts. In the end, the output of computations is decrypted with the algorithm. The output is 80 and 100 as expected.

```
Plaintext 1 : 73

Plaintext 2 : 20

Ciphertext ct1(73):

ct1_0: [21833 26574 13720 28250  7681 30423 17981 21471 22240  8624 19549  8058
   681 22645  9993 28528]
ct1_1: [27371 10088 12629 13280 21572 13524 30678  1191 22784  9988  1049  4577
 13068 16990  4151 29096]

Ciphertext ct2(20):

ct2_0: [20970 15209  4150  4589 18467 19280 21892 17046 28835 25313  7531 11470
 15900 29658 30115 17152]
ct2_1: [18596 25897  4387 29699  9356 13001 11953  6985  8229 24925 13711 20529
 11283  7941  6143 25565]

Encrypted ct3(ct1 + 7): (array([22729, 26574, 13720, 28250,  7681, 30423, 17981, 21471, 22240,
        8624, 19549,  8058,   681, 22645,  9993, 28528], dtype=int64), array([27371, 10088, 12629,
13280, 21572, 13524, 30678,  1191, 22784,
        9988,  1049,  4577, 13068, 16990,  4151, 29096], dtype=int64))

Encrypted ct4(ct2 * 5): (array([ 6546, 10509, 20750, 22945, 26799, 30864, 11156, 19694, 13103,
       28261,  4887, 24582, 13964, 17218, 19503, 20224], dtype=int64), array([27444, 31181, 21935,
17423, 14012, 32237, 26997,  2157,  8377,
       26321,  3019,  4341, 23647,  6937, 30715, 29521], dtype=int64))

Decrypted ct3(ct1 + 7): 80
Decrypted ct4(ct2 * 5): 100
```

Figure 3: Execution of Algorithm

5.2 Evaluation: Application of Fully Homomorphic Encryption for preserving privacy in a Machine Learning model(Linear Regression)

In order to evaluate the Homomorphic Encryption on Linear Regression model for predicting diabetes progression, we encrypt one of the Body Mass Index values and pass it to the model. The model predicts the diabetes progression and sends it back in encrypted format. We then compare the decrypted diabetes progression with the value obtained by predicting using the model in a non-encrypted format. We can see from the results that the values are identical up to six decimal places.
Next, we create two sets of prediction for test data set. First, with Homomorphic Encryption and the other without Homomorphic Encryption. We compare the two sets using R2 score. The R2 score obtained is 0.99. As the score is close to 1, the values are identical to each other. Thus, the error due to encryption is minimal. As the size of test set increases, error can increase marginally.

```
[13] # predict for bmi using encryption
     bmi1 = -0.0730303
     ctxt1 = HE.encryptFrac(bmi1)
     dp_encrypted = predict_encrypted(ctxt1)
     dp = HE.decryptFrac(dp_encrypted)
     print(dp)

     84.39906937489286

[14] # predict for the same value of bmi without encryption
     predict(-0.0730303)

     84.3990693477158
```

Figure 4: Predicting Diabetes Progression

6 DISCUSSION

To begin with, while working on this project, we have spent most of our efforts in understanding Homomorphic Encryption and its different types. We came across various limitations mainly in terms of number of computations of the Partially Homomorphic Encryption and Somewhat Homomorphic Encryption. We can also conclude from reading that Fully Homomorphic Encryption is most popular among all the types. We learnt through the initial implementation that basic mathematics and the concept of Learning with Ring Error is used to achieve Homomorphic Encryption.
Next, we applied Homomorphic Encryption in a Linear Regression problem to predict diabetes progression based on Body Mass Index. While working on this implementation, we came across different open source libraries developed for Homomorphic Encryption. Through the application we deduced two main points. First, Homomorphic Encryption can be used to successfully preserve privacy in sensitive data. Second, there is a possibility of error when Homomorphic Encryption is used. The amount of error can be managed by applying proper scheme of encryption.
One of the main challenges we are faced is applying Homomorphic Encryption to an entire data set instead of a single data entry. It is computationally expensive and extremely slow when applied to a data set. It is also too limited in terms of functionality for most uses. Hence, we used a small data set of 30 entries for our implementation.

6.1 Limitations and/or Future Work

Fully Homomorphic Encryption suffers from the limitation of not allowing multiple instances of the encryption scheme to be applied simultaneously. Hence, we cannot allow the encryption of two different data sets at the same time. Similarly, we cannot simultaneously have two different encryption-decryption schemes with separate keys for applying different Machine Learning models on the same data set. Thus, we are limited to a single-use-at-a-time framework that only allows the FHE scheme to be applied to singular instances of the data-set at any given moment.

Another limitation faced by the FHE schemes is that of extensive computational overhead faced by the system when handling complex mathematical operations (For instance, performing a reinforcement learning procedure through a ML tool such as Recurrent Neural Networks). In such a scenario, we need to perform a multitude of mathematical operations such as estimating the Activation function across multiple epochs. While these operations are computationally feasible on unprocessed data, the same operations become computationally infeasible when we perform them on encrypted data sourced through the FHE schemes. Hence, we need to find a way to be able to perform computational procedures of any complexity on the encrypted data provided by FHE schemes.

Hence, a suggested direction of academic research in the proposed approach would be to identify how we can implement Homomorphic Encryption based techniques on a larger scale by overcoming the challenges entailed by the extensive computational complexity of operations for the state-of-the-art Machine Learning models. The solution may lie within optimizing the Homomorphic Encryption techniques further to allow for more leeway in terms of computational power for performing the Machine Learning model operations.This would allow us to securely handle not only much larger data-sets but also more complex Machine Learning methodologies.

Additionally, we need to find a way to allow for simultaneously applying the FHE schemes across multiple threads. This would allow us to efficiently build secure, distributed Machine Learning pipelines at scale.

7 INDIVIDUAL CONTRIBUTIONS

All of us extensively collaborated in the initial-phase of our project to understand how the problem of preserving privacy within Machine Learning models needs to be handled and consequently came up with the idea of using Homomorphic Encryption as a solution.

Sankalp and Aditi researched extensively on the existing academic literature on the different variants of Homomorphic Encryption i.e. FHE, PHE, and SHE. The review was conducted with view towards getting a better picture of how Homomorphic Encryption has been used as a potential solution of our research problem, the state-of-the-art Homomorphic Encryption techniques being used, and the future research trends along which we could expect to contribute.

Once we had settled on Fully Homomorphic Encryption as a promising technique along which we could model our proposed solution, Shreya was responsible for putting together a initial framework through which we could build a Python-based implementation of our proposed FHE scheme. She was responsible for setting up the experiment and recording the various observations that could help us analyse the efficacy of our proposed model.

After we obtained a preliminary implementation of Fully Homomorphic Encryption in Python, We shifted focus on how we can apply the scheme to an actual Machine Learning model. Aditi worked on identifying an ideal Machine Learning technique that could be implemented on the encrypted data in a manner that provided results suitable for analysis and deriving conclusions that could scale and generalize to multiple popular Machine Learning techniques. Once we had identified Linear Regression as an ideal ML model, Aditi subsequently found a sensitive data set(Diabetes Data set [8]) that was suitable for analysis through our chosen Machine Learning model.

Once we had obtained the preliminary data and models required for our analysis, Aditi and Sankalp collaborated for implementing the Pyfhel-backed python implementation for our experiments. They worked together on building the source-code and setting up the various parameters for analysing the results. Subsequently, Shreya and Aditi were responsible for evaluating the obtained results, running additional tests and for drawing the relevant conclusions for presenting the outcomes for our proposed approach.

Finally, we collaborated as a team for analyzing the results that we obtained and for putting together the documentation for our report.

8 CONCLUSION

To summarize, we have researched about preserving privacy in the field of Machine Learning through Homomorphic Encryption. We studied the different types Homomorphic Encryption and various schemes of Fully Homomorphic Encryption. We also implemented one of FHE schemes in Python and tested it with basic arithmetic. In the end, we applied Homomorphic Encryption on Linear Regression model for diabetes data set.

Thus, we can conclude that our work is relevant in an era when the focus on privacy is increased, mostly because of regulations such as GDPR. The proposed approach through homomorphic encryption is promising for securely building real-world Machine-Learning applications across a variety of industries. Our proposed approach combines the need to protect privacy with the need to provide more detailed analysis through ML models, and hence takes us from a potential position of weakness to a potential position of strength, where we can securely build Machine Learning models without sacrificing the privacy of the data we use.

REFERENCES

[1] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press, 1978.

[2] Andrew Chi-Chih Yao. 1982. Protocols for secure computations. In FOCS, Vol. 82. 160–164.

[3] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009, ACM, 2009.

[4] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." ACM Transactions on Computation Theory (TOCT) 6.3.

[5] Brakerski, Z. (2012). Fully homomorphic encryption without modulus switching from classical GapSVP. In Advances in cryptology–crypto 2012 (pp. 868–886). Springer, Berlin, Heidelberg.

[6] Fan, J., Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, 2012, 144.

[7] Cheon, J. H., Kim, A., Kim, M., Song, Y. (2017, December). Homomorphic encryption for arithmetic of approximate numbers. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 409–437). Springer, Cham.

[8] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani (2003). Least Angle Regression

[9] Sklearn's Linear Regression https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[10] Ringing Learning with Errors https://en.wikipedia.org/wiki/Ring_learning_with_errors

[11] Lyubashevsky, V., Peikert, C., Regev, O. (2010, May). On ideal lattices and learning with errors over rings. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 1-23). Springer, Berlin, Heidelberg.

[12] Pyfhel: Python for Homomorphic Encryption libraries https://github.com/ibarrond/Pyfhel

9 PROJECT LINKS

9.1 GitHub Repository Link

https://github.ncsu.edu/scparik2/CSC533_Project
We have added two files to the above repository.

- Team14_HE.py: Python file which contains code for preliminary implementation and evaluation of Homomorphic Encryption Scheme
- Linear_Regression_Homomorphic_Encryption.ipynb: Python notebook which contains code for predicting diabetes progression using Linear Regression and Homomorphic Encryption

9.2 Dataset Link

https://www4.stat.ncsu.edu/ boos/var.select/diabetes.html
We used the Diabetes dataset for using Homomorphic Encryption on Linear Regression model.