International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 07 | July 2020 www.irjet.net p-ISSN: 2395-0072

Privacy-Preserving MLaas on Encrypted Data using Crypten

Dost Arora[1], Jibran Mukhtar Mattoo[2], Merin Meleet[3]

[1]Student, Dept. Information Science and Engineering, R.V. College of Engineering, Karnataka, India
[2]Student, Dept. Information Science and Engineering, R.V. College of Engineering, Karnataka, India
[3]Assistant Professor, Dept. Information Science and Engineering, R.V. College of Engineering, Karnataka, India

---

Abstract - *If one needs to apply any machine learning approach to a problem that requires some sensitive or private data such as medical or financial, it becomes imperative that close attention must be paid to privacy and security of data apart from its accuracy. This paper focuses discussion on a machine learning methodology that allows inference on encrypted data. This allows a data owner to send their data in an encrypted form to a cloud service that hosts the network. The encryption ensures that the data remains confidential since the cloud does not have access to the keys needed to decrypt it. It is achieved by using the protocol defined for FHE (Fully Homomorphic Encryption) and Secure MPC. In this paper, it is demonstrated that neural networks can be trained with the aid of encrypted data such that predictions can be retrieved in encrypted form. The research findings indicate that it provides appropriate training and classification that not only preserves privacy but is also efficient.*

Key Words: **Machine Learning, Privacy, Security, Homomorphic Encryption, Crytography, Multi-Party Computation, MLaaS**

1.INTRODUCTION

In spite of the growing popularity of machine learning prediction services in recent years, it is rare that the privacy of the consumer's information data is regarded as the top priority in their deployment. However, it is important to realize that we cannot eliminate this privacy requirement, whenever they want to apply machine learning to problems involving sensitive data. An example of such a case could be in medical applications. Suppose a hospital decides to accelerate its medical diagnosis using medical imaging. It could be that the hospital does not have sufficient labeled data to train statistical models. This could nevertheless work with a third-party service provider which would be able to provide these projections, but in so doing the hospital should satisfy all the ethical and legal criteria about patient information sensitivity. This paper discusses an approach that would allow hospitals to use such a valuable service without sacrificing patient privacy. In the protocol discussed, any private information could be encrypted by the hospital and then sent to the prediction service that operates in the cloud in an encrypted form. The cloud would then, be able to process the encrypted data and compute predictions over it, which would be sent back as results to the hospital that can decrypted and read by it.

Many machine-learning programmers are accustomed to using frameworks in Python language like Tensorflow, Scikit-learn, and PyTorch. These frameworks provide support for various high-level data types like real vectors, matrices, and many non-linear functions such as the logistic function and ReLu. For a modern MPC package, it becomes almost impossible for any data scientist to recreate and re-define all these frequently taken-for-grant primitives that are used with Machine-learning. Also, it is very expensive for any MPC expert to rewrite all the algorithms that they may require in a machine learning application. It is therefore important to design a MPC front end that is simple and convenient to today's Python and PyTorch community. In fact, many machine learning frameworks use Python as a frontend language and offer Numpy-style array operations to simplify machine learning.

This paper studies such a framework called CrypTen (by Facebook) and demonstrates its effectiveness in protecting the privacy of users by employing techniques of collaborative machine learning and statistical inference. We do so by studying various performance metrics involved in data analysis and inference gathering such as model accuracy, model training time, the robustness of encryption, etc. It is further shown that it delivers an intuitive end-to-end solution that enables various Python interfaces that are similar to those from PyTorch, one of the most popular Python packages, as well as a variety of functions frequently used in machine learning.

2.RELATED WORK

In regard to machine learning, there are also other non-cryptography-based approaches that preserve privacy. In [1], Shokri and Shmatikov propose an approach that allows the neural network model to learn from a distributed dataset that enables multiple parties to preserve their privacy by refraining from sharing their input datasets. The main aim of the paper was to present an approach that can allow the data to be decentralized so that various datasets which involve sensitive information like user's personal photos, voice recordings are not required to be uploaded to any central repository for it to be consumed by a machine learning model. Their idea is built on differential privacy, where each party uses its data

to build the model and only shares a minor fraction of the parameters with other parties. Their key innovation is to employ selective sharing of the model parameters. Since the stochastic gradient descent algorithm can be parallelized and can run asynchronously, it is feasible for participants who are collaborating in the training stage to benefit from other participants' models, without sharing their inputs explicitly.

In [2] Abadi et al. propose a framework for differentially private training of neural networks. This framework includes a stochastic gradient descent algorithm that is differentially private in nature and hyperparameter tuning. In doing so they develop new algorithmic techniques and implementation strategies using the TensorFlow library and further provide an analysis of privacy costs involved within the framework of differential privacy. It has been observed that differential privacy-based approaches are required to make a trade-off between the privacy of the model and its utility which in turn affects its accuracy. Our model of threat is distinct to that of differential privacy. The server doesn't really learn anything at all about the model in our methodology whereas the server will learn the model in differential privacy. Reverse engineering cannot be used either, because the code is encrypted. Since the model is encrypted, reverse engineering cannot be used either.

In [3], the authors develop a framework that uses a Fully Homomorphic encryption scheme based on HElib[4]. Such frameworks seem to be slow because of the Heavy computation involved. Using this framework they infer various statistics including Histogram count, k-percentile, principal component analysis, contingency table, etc. They perform an empirical analysis of these statistics on various categories of datasets, including categorical data, ordinal data, and numerical data, belonging to UCI Machine Learning Repository [5].

In [6], the authors develop a multi-leveled framework that exposes various primitives of SEAL [7] to TensorFlow. Here SEAL serves as the encryption layer and provides various encryption protocols in the context of Fully Homomorphic Encryption which includes schemes such as BFV[9] and CKKS[8]. Such a framework is similar to CryptoNets[10] model, but the authors are able to improve the inference time of the machine learning model by speeding up expensive operations such as squared activation function by sparsifying the network activations of the model as described in [11]. The authors also introduce a novel approach of encrypted transfer learning that allows the client to query a cloud that can expose embeddings obtained from variational autoencoder learning of unlabelled dataset such that it can be transferred to client downstream.

3.BACKGROUND

MPC has been studied for many decades, but recently complicated calculations have become feasible because of newly discovered enhancements in implementation such as the availability of faster hardware and extremely simple AES block cipher deployments and the exposure to easy networking. Circuits can now be evaluated at up to 7 billion gates per second, provided that the measurement is viewed as a Boolean circuit. MPC as a cryptographic paradigm allows for computational collaboration. Nevertheless, MPC suffers from drawbacks for the purposes discussed in this study, such as, it is still infeasible to run generic MPC protocols over many remote parties especially due to high communication costs.

The library used is a PyTorch-based machine learning platform that allows you to quickly test and create machine learning methods using safe computer technology. It offers the ability, along with and similar to the PyTorch API, to create models when measuring encrypted data without exposing the sensitive data. It is ensured that confidential or otherwise private information stays private while enabling model inferences and training on encrypted information that can be aggregated between different organizations or users.

The framework utilizes secure multiparty computation (MPC) as its cryptographic paradigm. Secure multi-party computation (MPC) enables users to evaluate a function without disclosing confidential information other than the outcome. MPC often uses several cryptographic techniques with different priorities of efficiency and protection, e.g. disordered circuits and secret sharing. After more than 30 years of growth, one started seeing real-world data processing applications start using MPC. Although there are still some challenges hindering the mainstream use of efficient computing techniques.

3.1 Secure Multi-Party Computation (MPC)

Assume that there are k parties such that each party i holds a private value x_i. The goal is to evaluate a function $f\ (x_1; \dots\ ;\ x_k)$ so that each party will learn the result, but no information about each other's inputs x_i beyond what can be inferred from the result. Some MPC techniques are generic in the sense that they can be applied to a broad range of functions (such as Boolean circuits), while other techniques may be designed for a specific set of functions

3.2 Secret Sharing

Secret sharing is designed to allow the storage of data to be distributed so that no information about each share can be discovered but that the entire data can be recovered when combined. For example, assume x is a number in the range $[C; C)$. One can pick a random number r uniformly sampled from the same range and

create a secret share such that in one location r is recorded, and in another location $x + r \bmod C$ is recorded. Since both r and $x + r$ are uniformly distributed in $[C; C)$, each one of them in isolation does not reveal any information about x. However, having access to both allows computing the secret x. It is easy to see that two such secret shares can be added to create a secret share of the sum of the secrets. Therefore, the type of secret sharing presented here provides an efficient method to implement an MPC protocol for the addition function.

3.3 Security of MPC

A number of MPC security definitions have been put forward [12], and they aim to ensure a number of key security characteristics that are generally sufficient to capture the most (if not all) multi-party computation tasks. The most important of these properties are now defined. In so doing let's take an example where Alice, Bob, and Charlie want to find out what their highest wages are without disclosing their respective wages.

1. Privacy: No party must discover more than its outcome authorized. In particular, only information about the inputs of others should be learned from the output itself. In the above case, where the only highest pay is reported, it is apparent that all other wages are lower than the highest. Nonetheless, nothing more about the other wages should be known.
2. Correctness: The correct output is assured for each party. Continuing with the instance of salaries indicates that the individual with the highest wage is expected to benefit.
3. Guaranteed Output Delivery: Corrupted parties cannot stop honest parties from getting their results. In other words, by performing a DOS attack, the attacker won't be able to disable the system.
4. Fairness: Corrupted parties should receive their results only if they receive their results. The scenario in which a corrupt party gets output and an honest party should not happen. For eg, in the event of contract signing, this property can be crucial. Specifically, the possession of the signed contract by the dishonest party would be very problematic if it is not present with an honest party. Note that guaranteed performance implies fairness of computation, but not necessarily the converse.
5. Independence of Input: Corrupted parties must select their input regardless of the honest parties' input. In scenarios like secured bidding where bids are held, and parties have to set their bids independently of others, this property becomes critical. It should be noted that input independence does not mean privacy.

The above definition of protection doesn't answer one very critical aspect: the strength of the adversary targeting protocol execution. The adversary controls a sub-set of the parties participating in the protocol as it was said. This approach takes on the "honest but curious" threat model. This is sometimes referred to as the adversary's semi-honest model. The following assumptions are made on the Threat Model:

1. Every party is following the protocol faithfully: i.e., it carries out all of the programs and communicates the correct results to the parties specified in the program.
2. The channel of communication is safe: no party can view any data that is not communicated directly with it.
3. Every party has access to a private source of randomness, e.g., a private coin to toss
4. Parties may use any data they have already seen and perform arbitrary processing to infer information.

4. TECHNICAL DETAILS OF MPC

To implement the MPC protocol for an honest majority, one would require a secret sharing protocol as a basic tool. A secret sharing scheme allows a dealer to share a secret s among the parties so that any subset of $t + 1$ parties can reconstruct the secret from their private shares, yet no other subset of size less than $t + 1$ can learn anything about the secret. As discussed earlier such a scheme that fulfils the requirement is called $(t + 1) - out - of - n$ threshold secret sharing scheme. Shamir's secret scheme is such an example.

4.1 Shamir's Secret Sharing

Shamir's secret sharing scheme starts by building a polynomial on a two-dimensional plane. It is known that a polynomial of degree at most t, can be uniquely identified by t pairs of unique coordinates. After obtaining $t + 1$: $(x_1, y), \ldots, (x_{t+1}, y)$ with unique x_i , then it implies that there exists a unique polynomial $q(x)$ such that $q(x_i) = y_i$ for every i. Furthermore, it also implies that it would be possible to find image $q(x)$ of any point defined with its ordinate as x. One way to construct such polynomial is via Lagrange basis polynomials $l_1(x), \ldots, l_t(x)$, in which reconstruction is carried out by computing $q(x) = \sum_{i=1}^{t+1} l_i(x) \cdot y_i$.

Suppose all computations are performed in the finite field Z_p, for a prime $p > n$. Now, if a dealer wants to share a secret s, they would have to choose a random polynomial of degree at most t while keeping the constraint that

$q(0) = s$. Specifically, the dealer chooses $a_0 = s$ and chooses other random coefficients $a_0, \ldots, a_t \in Z_p$, and sets $q(x) = \sum_{i=0}^{t} a_i \cdot x^i$. To distribute it, the dealer would then provide a share $y_i = q(i)$; to the $i'th$ party for every $i = 1, \ldots, n,$. This is the reason why it is required that $p > n$, so that different shares can be given to each party. If the parties want to reveal the secret share they can do that by reading the value for $q(0)$. Any t parties out of $t + 1$ can reconstruct the polynomial by simply interpolating the polynomial, by using their share to eventually compute $q(x)$. Also, it is shown that any subset of t or fewer parties cannot learn anything about s. This is due to the fact that they at least t are needed to reconstruct a unique polynomial and there exists a polynomial, going through t or fewer points and the point $(0, s)$ for every possible $s \in Z_p$. Furthermore, since the polynomial was chosen at random, all polynomials are equally likely, and so all values of $s \in Z_p$ are equally likely.

4.2 Key Sharing Set-Up

In most of the MPC protocols, the primary step to compute any function is to represent that function as a circuit, either in Boolean form or in arithmetic, as per the requirement. These circuits are comprised of primitive multiplication and addition gates. To perform MPC in an honest-majority scenario all the arithmetic circuits are constructed over a finite field Z_p with $p > n$. One should note that these circuits are Turing-complete, which implies that in theory any function can be represented using these circuits. The protocol for semi-honest adversaries consists of the following phases:

1. Input sharing: In this phase, each party would utilize Shamir's secret sharing to share their private input to all other parties. Each party would play as a dealer for their associated input wire, thereby sharing the input to all the other parties. In doing so, not a subset of minority parties will ever be able to learn anything about the shared values. Following this step, the parties hold secret shares of the values on each input wire.
2. Set-Up: The set-up procedure uses the SPDZ protocol [13]
    a. A secret key $\alpha \in \mathrm{F}_p$ is chosen globally for all the n parties partaking in collaborative computation
    b. Each party holds a share α_i of global key α such that:

$$\alpha = \alpha_1 + \cdots + \alpha_n$$

Here all the data will be represented by elements of F_p. If a dealer wants to share a secret value $x \in \mathrm{F}_p$ among the parties, they can share as follows:

1. Each party i will hold a share of data x_i.
2. Furthermore, each party i will also hold a share of "MAC" share $\gamma_i(x)$

Such that,

$$x = x_1 + \cdots + x_1$$
$$\alpha \cdot x = \gamma_1(x) + \cdots + \gamma_1(x)$$

From now on, $[x]$ will be used to denote such a sharing of x. Before any further computations can be performed, some pre-processing is needed to be performed, therefore it is said that the protocol works in the pre-processing model. The result of this preprocessing is independent of the data on which it will be applied or the function it will be utilized.

Three triples of shared values are generated: $[a], [b], [c]$, called Beaver Triplets [14]. This is achieved through a "Resharing Scheme" which involves the implementation of Fully Homomorphic Encryption Scheme, such that a public key pk is shared among parties. For this, any Homomorphic scheme which uses LWE assumption, such as Brakerski–Vaikuntanathan scheme [15] can be used. Each party i holds share sk_i of the secret key. They can come together to decrypt any ciphertext ct, $Dec_{sk_1 \ldots sk_n}(ct)$.

The Resharing-Scheme involves when a given ciphertext ct encrypting the value m has to be shared so that each party can obtain m_i, such that $m = \sum m_i$. It involves the following steps:

1. Party i generates a random f_i and transmits $ct_{f_i} = Enc_{pk}(f_i)$
2. All compute $ct_{m+f} = ct + \sum ct_{f_i}$
3. Each execute $Dec_{sk_1 \ldots sk_n}(ct_{m+f})$ to obtain $m + f$
4. Party 1 sets $m_1 = (m + f) - f_1$
5. Party $i \neq 1$ sets $m_i = -f_i$

For the generation of $[a]$ and $[b]$, the following technique is utilized:

1. Party i generates a random a_i and transmits $ct_{a_i} = Enc_{pk}(a_i)$
2. All compute $ct_a = \sum ct_{a_i}$
3. All compute $ct_{\alpha \cdot a} = ct_\alpha \cdot ct_a$
4. Execute Reshare on $ct_{\alpha \cdot a}$ so party i obtains $\gamma_i(a)$

For generating $[c]$, follow:

1. All compute ct_c from ct_a and ct_b
2. Get shares c_i via executing Reshare-Scheme on ct_c; also obtaining a $ct_c^{`}$
3. All compute $ct_{\alpha \cdot c} = ct_\alpha \cdot ct_c^{`}$

4. Execute Reshare-Scheme $ct_{\alpha \cdot c}$ so party i obtains $\gamma_i(c)$.

4.3 Arithmetic Operations

Here, an interesting concept is employed, that every operation that needs to be carried out can be expressed by gates of addition and multiplication. In other words + and × are a set of Universal Gates over F_p. The inputs from a party are shared using the above-discussed scheme.

1. Addition:
   Suppose two values $[x]$ and $[y]$ are shared. And one needs to perform an additional operation to get the result $[z]$. To do so, the parties will do execute the following individually:
   1. $z_i = x_i + y_i$
   2. $\gamma_i(z) = \gamma_i(x) + \gamma_i(y)$

$$Z = \sum Z_I = \sum (x_i + y_i) = \left(\sum x_i\right) + \left(\sum y_i\right) = x + y$$

$$\alpha \cdot z = \sum \gamma_i(z) = \sum \left(\gamma_i(x)\right) + \left(\gamma_i(y)\right) = \alpha \cdot x + \alpha \cdot y = a \cdot (x + y)$$

From the above result, it is evident that any linear function can be computed. Example:

Given constants v_1, v_2, v_3 and shared values $[x]$ and $[y]$ one can compute

$$v_1 \cdot [x] + v_2 \cdot [y] + v_3 = [v_1 \cdot x + v_2 \cdot y]$$

These values will be used to evaluate multiplication results.

1. Multiplication:

Note: Partially opening a share $[x]$ means revealing x_i but not the MAC share.

Suppose one needs to evaluate $[z]$ from $[x]$ and $[y]$:

1. Choose a new triple ($[a]$, $[b]$, $[c]$) from the precomputed list.
2. Partially open $[x] - [a]$ to obtain $\epsilon = x - a$
3. Partially open $[y] - [b]$ to obtain $\epsilon = y - b$
4. Now compute the following function locally:

$$[z] = [c] + \epsilon \cdot [b] + \rho \cdot [a] + \epsilon \cdot \rho$$

Note: Each multiplication requires interaction. If a is random then ϵ is one-time pad encryption of x.

$$[c] + \epsilon \cdot [b] + \rho \cdot [a] + \epsilon \cdot \rho = \mathrm{a} \cdot b + (x - a) \cdot b + (y - b) \cdot a + (x - a)(y - b)$$

$$= (a \cdot b) + (x \cdot b - a \cdot b) + (y \cdot a - a \cdot \mathrm{b}) + (\mathrm{x} \cdot y - x \cdot b - y \cdot a + a \cdot b) = x \cdot y$$

5. METHODOLOGY

We will go through these 3 categories of use-cases for the framework being discussed.

1. Feature Aggregation:
   In the first case, several parties have different feature sets and want the joint feature set to be measured without data sharing. For example, various providers of health care may each have a medical history but want to use the entire medical history of the patient to make accurate decisions while also safeguarding patient privacy.

2. Data Labeling:
   There, one party holds feature data, while another party holds the correct labels, and all the parties want to learn a relationship without exchanging data. This is similar to the addition of features except that one party has labels rather than other features. Suppose, for example, that one healthcare provider had access to clinical results data in previous healthcare cases, while other individuals had access to health apps. The parties may want to build a model that predicts health outcomes based on characteristics without disclosing any information on health between parties.

3. Model Hiding:
   One party has access to a trained model in this scenario, whilst another party wants to use that model for its data. The data and model must, however, be kept private. This can happen when a model is proprietary, costly to manufacture, and/or vulnerable to white-box attacks but has value for more than one party. In the past, the second party has had to send the data to the first to implement the model, but privacy-preserving methods can be used if the data is required to be kept safe and not exposedWe tested the ML models on the MNIST dataset

This dataset consists of 60,000 images of handwritten digits. Each image is a pixel array having dimensions of 28x28. Here each pixel is represented by a number in range 0-255 denoting its grey level. 50,000 images were used as the training part of the dataset, 10,000 images for testing.

5.1 Feature Aggregation & Data Labelling

For the use-case of feature aggregation and data labeling, a neural network was built that distinguishes between zero and non-zero number images. Here the feature dataset was split in such a way, that it results in Alice having the first 28 x 20 features and Bob having the last 28 x 8 features. One way to think of this split is that Alice has the (roughly) top 2/3rds of each image, while Bob has the bottom 1/3rd of each image. Alice and bob collaborate in training on the dataset by establishing a communication channel through MPI. This interface is

needed for the framework to communicate among the parties. Consequently, the world-size parameter for MPI has been kept as 2. The training is run for 10 epochs, each for Alice & Bob. Here the batch size is kept as 16. The learning rate was kept as 0.001. The network summary is as follows:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 24, 24]             416
            Linear-2                [-1, 100]         230,500
            Linear-3                  [-1, 2]             202
================================================================
Total params: 231,118
Trainable params: 231,118
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.07
Params size (MB): 0.88
Estimated Total Size (MB): 0.96
----------------------------------------------------------------
```

Fig-1: Model Architecture

1. Convolution Layer: The input image is 28 * 28. The convolution has windows, or kernels, of size 5 * 5, and a padding of 0.
2. ReLu Activation Function: This layer applies the ReLu function to each of the incoming values from the previous layer.
3. Max-Pooling: Performs a max-pooling and reduces the size from 24 * 24 to 12 *12
4. Linear Layer 1: Applies a linear transformation to the incoming data of batch size 16, each of dimension 12 * 12 and produces an output of dimension 100.
5. Linear Layer 1: Applies a linear transformation to the incoming data of batch size 16, each of dimension 1*100 and produces an output of dimension 2.

An accuracy of 93% was achieved.

5.2 Model Hiding

In this scenario, Alice is having a pre-trained model with all its weights and biases stored in the disk. Bob would like to perform classification on the MNIST dataset, without revealing the dataset to Alice. The approach here would be to encrypt the weight and biases on Alice, along with Bob also encrypting his dataset tensor. Let's start with a simple neural network with the following architecture.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 128]         100,480
            Linear-2                [-1, 1, 128]          16,512
            Linear-3                 [-1, 1, 10]           1,290
================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.45
Estimated Total Size (MB): 0.46
----------------------------------------------------------------
```

Fig-2: Model Architecture

This model serves as a dummy model for Alice, into which the pre-trained model can be initialized. Loading a model will assert the correctness of the model architecture provided against the model loaded. A dummy input would also be needed, which should have the same shape as the model expects the input tensor to have. Here the dummy input is chosen as a tensor of shape $[1, 784]$. The framework allows the dummy input to seep through every layer of the model and encrypt all the weights and biases that it comes across. This results in a yield of output tensor whose values have been encrypted, with a size of $[10000, 10]$. Furthermore, an accuracy of 96% is achieved.

6. BENCHMARKING

6.1 Runtimes

Here, various runtimes of mathematical functions are studied. As discussed, the MPC can perform any computation that can be represented by a series of multiplication and addition. In other words, + and × multiplication forms the set of universal gates for F_p. Therefore, every complicated mathematical function has to be represented in addition or multiplication. For instance, the square root function can be implemented using Newton-Raphson's method of approximation. Following observations are madebench.

1. Function runtimes are averaged over 10 runs using a random tensor of size $(100, 100)$.
2. max and $argmax$ are excluded as they take considerably longer

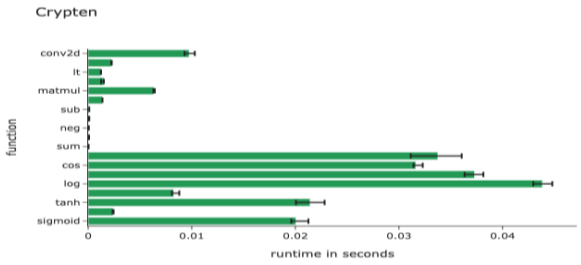

Fig-3: Crypten Mathematical Function Runtimes

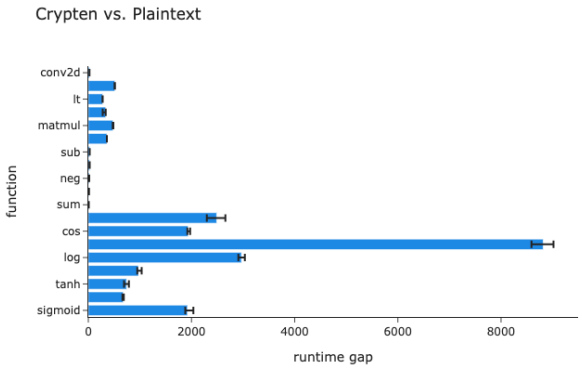

Fig-4: Crypten vs Plaintext Runtimes

6.2 Errors

1. Function errors are over the domain $(0, 100]$ with step size 0.01
2. *exp*, *sin*, and *cos* are over the domain $(0, 10)$ with step size 0.001

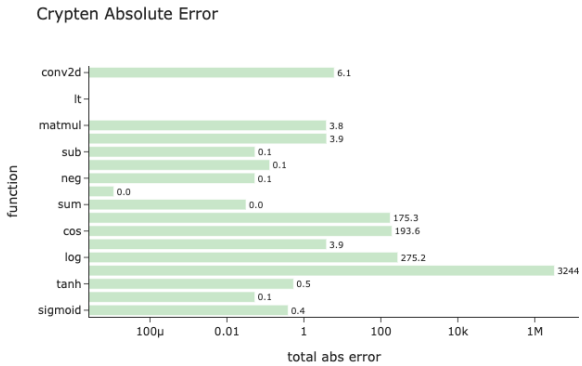

Fig-5: Crypten Mathematical Function Absolute Errors

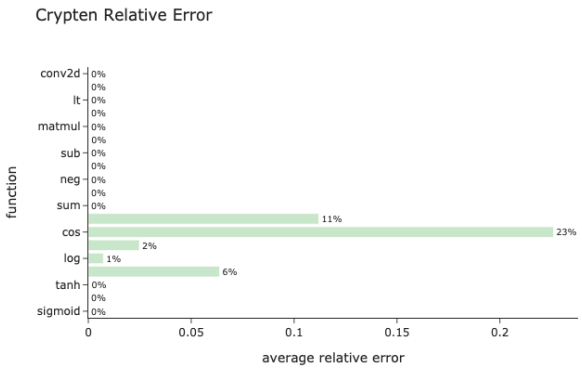

Fig-6: Crypten Mathematical Function Relative Errors

6.2 Models

We trained the models on Gaussian clusters for binary classification. It Uses SGD with 5k samples, 20 features, over 20 epochs, and 0.1 learning rate. The Feedforward has three hidden layers with intermediary RELU and final sigmoid activation.

As we can observe in the result, that model training time is considerably high, with as far going to 14 times, but the real advantage lies in training it on encrypted dataset. This inference on encrypted dataset yields an accuracy that is similar to that of plaintext.

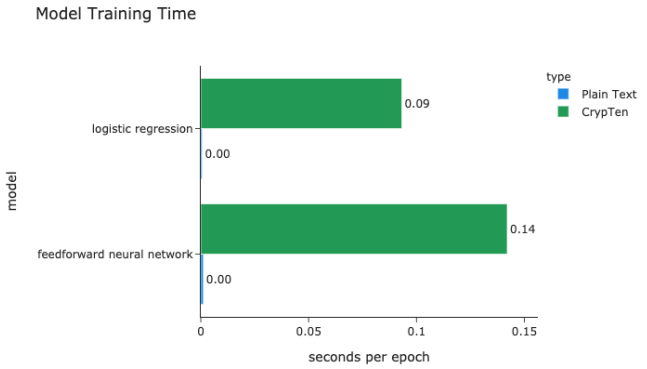

Fig-7: Crypten Model Training Time

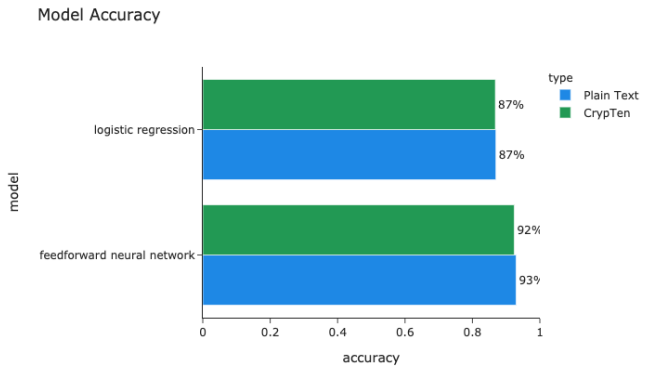

Fig-8: Crypten Model Accuracy

It can be concluded here that, although Crypten makes relative errors in computation, most prominently in trigonometric approximations of *sin* and *cos*, these errors get diluted in the overall inference of the model, thereby maintaining a similar accuracy as plaintext models.

7. CONCLUSION

Cloud computing introduces a new approach to information management and retrieval. The cloud comprises a massive, elastic, powerful pool of computer networks. Cloud services include software as a service (Saas), platform as a service (Paas), and infrastructure as a service (Iaas). Services offered by cloud providers include: These cloud-based computing infrastructures offer such machine-to-deep learning solutions as services that enable systems to be scalable, accessible, and maintained. Unfortunately, such an approach involves the treatment of sensitive data, e.g. personal images or videos, medical diagnosis, and data that may reveal ethnic backgrounds and political opinions but also genetic, biometric and health data.

The protection of consumer data privacy in the machine-learning as-a-service computing environment includes the use of encryption mechanisms such as homomorphic encryption and MPC, allowing users to work with encrypted data. Users would locally encrypt their data using a public key in such a way that they can send them to an appropriate Cloud-based machine learning provider and receiver of the authenticated computing results that are decrypted locally through their private key. Specifically, such architecture enables encryption and decryption phases performed on the user device from the processing performed by the Cloud-based computing infrastructure to be decoupled. Such a HE centralized architecture protects data protection while maintaining scalability, flexibility, and high efficiency of a Cloud-based system.

8. REFERENCES

[1] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). Association for Computing Machinery, New York, NY, USA, 1310–1321. DOI:https://doi.org/10.1145/2810103.2813687.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 308–318. DOI:https://doi.org/10.1145/2976749.2978318

[3] LU, W., KAWASAKI, S., AND SAKUMA, J. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. IACR Cryptology ePrint Archive 2016 (2016), 1163

[4] Halevi S., Shoup V. (2014) Algorithms in HElib. In: Garay J.A., Gennaro R. (eds) Advances in Cryptology – CRYPTO 2014. CRYPTO 2014. Lecture Notes in Computer Science, vol 8616. Springer, Berlin, Heidelberg

[5] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[6] van Elsloo, T., Patrini, G., & Ivey-Law, H. (2019). SEALion: a Framework for Neural Network Inference on Encrypted Data. arXiv preprint arXiv:1904.12840.

[7] Laine, K., & Player, R. (2016). Simple encrypted arithmetic library-seal. Technical report, Technical report.

[8] Cheon J.H., Kim A., Kim M., Song Y. (2017) Homomorphic Encryption for Arithmetic of Approximate Numbers. In: Takagi T., Peyrin T. (eds) Advances in Cryptology – ASIACRYPT 2017. ASIACRYPT 2017. Lecture Notes in Computer Science, vol 10624. Springer, Cham

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

[10] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., & Wernsing, J. (2016, June). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In International Conference on Machine Learning (pp. 201-210).

[11] Louizos, C., Welling, M., & Kingma, D. P. (2017). Learning Sparse Neural Networks through L_0 Regularization. arXiv preprint arXiv:1712.01312.

[12] Lindell, Yehuda. "Secure Multiparty Computation (MPC)." IACR Cryptol. EPrint Arch., vol. 2020, 2020, p. 300.

[13] Damgård I., Pastro V., Smart N., Zakarias S. (2012) Multiparty Computation from Somewhat Homomorphic Encryption. In: Safavi-Naini R., Canetti R. (eds) Advances in Cryptology – CRYPTO 2012. CRYPTO 2012. Lecture Notes in Computer Science, vol 7417. Springer, Berlin, Heidelberg

[14] Beaver D. (1992) Efficient Multiparty Protocols Using Circuit Randomization. In: Feigenbaum J. (eds) Advances in Cryptology — CRYPTO '91. CRYPTO 1991. Lecture Notes in Computer Science, vol 576. Springer, Berlin, Heidelberg

[15] Brakerski Z., Vaikuntanathan V. (2011) Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: Rogaway P. (eds) Advances in Cryptology – CRYPTO 2011. CRYPTO 2011. Lecture Notes in Computer Science, vol 6841. Springer, Berlin, Heidelberg