Irving Fisher Committee on
Central Bank Statistics

BIS

IFC-Bank of Italy Workshop on "Data science in central banking"

18-20 February 2025

Regression on encrypted datasets using fully homomorphic encryption[1]

Adriano Baldeschi and Giuseppe Bruno,
Bank of Italy

Mohammad Raeini, Paul Master and Vineet Chadha,
Cornami

[1] This contribution was prepared for the workshop. The views expressed in this publication are those of the authors and do not necessarily represent the official views of the Committee, its members, or the BIS.

Regression on Encrypted Datasets using Fully Homomorphic Encryption

Adriano Baldeschi[1], Giuseppe Bruno[1], Mohammad Raeini[2], Paul Master[2], Vineet Chadha[2]

[1] Banca d'Italia
[2] Cornami, Inc.

The views expressed in the studies are those of the authors and do not involve the responsibility of the bank of Italy.

Abstract. Fully Homomorphic Encryption (FHE) is a branch of modern cryptography that offers solutions for privacy in today's data-driven world. FHE is considered the holy grail of cryptography and data privacy. In the last couple of decades, there has been significant progress in FHE and it has been standardized to provide encrypted data solutions. Several FHE software packages and libraries have implemented different FHE schemes [BGV][CKKS][tFHE]. In particular, the CKKS FHE scheme has attracted significant attention as it enables FHE operations on encrypted floating-point numbers which are representative of real-world data while performing computation on encrypted data.

In this paper, we use the CKKS scheme on a common data analytics application, training a linear regression on FHE encrypted data. Then, we use the trained linear regression model to make inference on encrypted data obtained from the Private Set Intersection (PSI) of two datasets. We implemented the popular gradient descent algorithm using FHE primitives. The results of our experiments show that the hardware-accelerated implementation of the FHE primitives speeds up the encrypted gradient descent algorithm significantly.

Keywords: Fully Homomorphic Encryption, Hardware Acceleration, Linear Regression, Private Set Intersection, FHE, PSI, Gradient Descent.

1 Introduction

Digital data plays an important role in the modern world and brings significant value to society and the global economy. Various companies rely on data available via cloud platforms for multi-party computation data analytics applications. However, sharing data, particularly in an un-encrypted form on the cloud platforms, has resulted in increasing data privacy concerns. As our world becomes more and more data-driven, data privacy is becoming more important. Fortunately, since the early days of digital data, researchers have been working on solutions and mechanisms to protect digital data. Cryptographic techniques, such as FHE, offer promising solutions for addressing data privacy concerns.

FHE is a branch of modern cryptography that offers solutions for the increasing data security/privacy concerns of the modern data-driven world. FHE was first envisioned in the early

years of modern cryptography by the pioneers of cryptography [1]. While the envisioned idea remained an open problem for several decades, in 2009 the first promising solution was proposed [2] that took this domain of modern cryptography to the next phase of its evolution. Since 2009, there have been several FHE schemes (e.g., BFV, BGV, TFHE, CKKS) some of which have been proposed for standardization [3]. The FHE schemes are typically categorized into four generations. Among these schemes, the CKKS scheme [4] has attracted more attention, because it enables FHE operations on encrypted floating-point numbers which are representative of real-world data.

In this paper, we analyze the HW-acceleration of FHE schemes, by applying the CKKS scheme on a real-world data analytics task, training a linear regression on FHE encrypted data. For training a linear regression model in encrypted form, we implement the gradient descent algorithm using FHE primitives. We also use an FHE-trained linear regression model to make inference on the encrypted data. For this, we use the PSI technique to obtain the intersection of two datasets. The HW-accelerated implementation and results show a significant speedup of the FHE primitives that are needed for real-world data analytics applications like linear regression on encrypted data and for privacy-preserving machine learning (PPML).

1.1 Problem Statement

In this paper, we focus on a multiparty data analytics application, specifically, training a linear regression model on FHE encrypted data and then infer using the trained linear regression model on the intersection of two datasets. The two datasets are provided by two different parties that are not willing to share their data with each other.

Linear regression is a classical statistical approach. There are different approaches for training a linear regression model to a given dataset. Two common approaches are traditional linear regression (Matrix inversion) and stochastic gradient descent. We use stochastic gradient to implement in an encrypted format using FHE primitives. A brief explanation of this approach is explained in Section 3.1 of this document. In general, inference with a trained linear regression model is straightforward. It is essentially evaluating a linear function on a set of data points. While training a linear regression model on a plaintext dataset is well-studied, training a linear regression on FHE encrypted data has only been recently investigated. In this paper, we study training a linear regression model on a dataset, where the data points are encrypted with an FHE scheme.

Our goal was to implement the gradient descent algorithm using FHE primitives and to analyze its performance in two settings, i.e., software-only implementation and hardware-accelerated implementation.

This paper is organized as follows. In the next section, we review the related work in context of encrypted FHE applications. In section 2, we discuss architecture of our implementation. In section 3, we provide the details of our implementation for the gradient descent algorithm on encrypted data. In section 4, we present the results. Specifically, we show the software-only performance results for training a linear regression model on encrypted data as well as performance results for

making inference using the trained model. We also provide hardware-accelerated performance results (in section 4.2). In section 5, we provide our conclusion.

1.2 Related Work

Logistic and linear regression are two of the popular methods that have been utilized extensively for data analytics purposes. There have been numerous attempts for privacy-preserving evaluation of these two regression methods. In the following, we provide a brief overview of some of the previous works in literature. We mainly focus on homomorphic evaluation of linear regression as it is the topic of the current article.

The homomorphic evaluation of linear regression has been studied in several previous works, e.g., [5], [6], [7], [8]. The reference [5], which seems to be one of the recent attempts, used the SEAL[1] fully homomorphic encryption library for evaluation of regression model. The work provided general information about the scheme but did not provide the details of the proposed technique. On the other hand, to the best of our knowledge, the first attempt for designing FHE-based solution for linear regression was proposed in [8]. The proposed scheme [8] relies on Cramer's rule for computing the inverse of $X^T X$ which is required for training a linear regression model. According to the existing literature, the matrix inversion approach is more suitable for situations where the data has a low dimension (dimension < 6). Therefore, for high dimensional data researchers have tried other approaches.

In [7], the authors have used an iterative method for computing the inverse of matrices which are used for training a regression model. The proposed approach in [7], in turn, relies on a division free technique for computing the inverse of a matrix [9]. The authors in [7] have implemented their proposed technique using the C++ implementation of BGV FHE scheme [10] from HElib[2]. According to [7], the proposed technique was able to train a linear regression model on a dataset with 40k instances and at most 6 features in about 15 minutes. Note that the matrix inversion approach can be suitable only for datasets with a small dimension (dimension < 6).

Another attempt for homomorphic evaluation of linear regression was conducted in [6]. In this work, the authors studied the gradient-based approaches for training a linear regression model. In particular, they performed experiments regarding the applicability of Nesterov's accelerated gradient algorithm on data encrypted using the FV fully homomorphic encryption scheme [11]. The results in [6] indicate that an approach using the van Wijngaarden transformation (VWT) seems to work better than the Nesterov's accelerated gradient (NAG) algorithm. Whereas for the unencrypted case, the NAG algorithm is among the state-of-the-art algorithms.

There are also some previous related works that rely on homomorphic encryption schemes (not necessarily FHE) and some that use gradient descent algorithms for other optimization problems. For example, in [12] researchers have used the Paillier encryption scheme besides an iterative

[1] https://github.com/microsoft/SEAL
[2] https://github.com/homenc/HElib

approach for computing matrix inverse to design schemes for secure evaluation of linear regression. In [13], the authors have used (accelerated) gradient descent algorithms for solving quadratic programming optimization problem on encrypted data. They have analyzed the applicability of different descent algorithms with a few FHE schemes from Microsoft SEAL library; and concluded that the CKKS scheme is the only applicable scheme for implementing gradient descent algorithms [13].

2 Architecture: Design and Tradeoff

Figure 1 shows the architecture of integrated training and inference for multiple parties. An example use case for this architecture is two banks sharing their data to enable analytics on common data to identify fraudulent transactions. Our goal is to provide end-to-end security and a scalable approach for multiple features per data set and for large number of data sets. In summary, the implementation includes the following choices:

- We chose a CKKS based FHE scheme because it is representative of real time data.
- Multiple parties provide their data for common data analytics. Each party keeps their FHE keys private (it is never shared with anyone).
- Regression training and inference is performed in a cloud server.
- The goal was to speed up the solution using Cornami hardware acceleration. We used SCIFR [Reference] APIs to support multiple FHE frameworks. SCIFR seamlessly allows access to multiple frameworks.
- We implemented our solution using a 3rd party FHE layer (HElayers[3] from IBM) that supports multiple frameworks.
- The SCIFR APIs[4] for FHE allows the use of any FHE or meta FHE schemes.

We follow a multiparty computation setting, in which there are two clients (party A and party B) and a cloud server. Each of the two clients has a dataset, where each client wants to keep its dataset private. The goal is to train a linear regression model on a party's dataset (e.g., party A's dataset); and to make inference on the intersection of the two clients' datasets.

The two clients use threshold fully homomorphic encryption that allows them to use a joint public key to encrypt their datasets. The joint public key is used to encrypt the data. However, to decrypt the data, specifically the prediction result, both of the parties must contribute to the decryption process.

The training process works as follows:

To train a linear regression model on party A's dataset, party A encrypts its dataset using the joint public key. Party A then sends its encrypted data to the cloud server. The cloud server uses a

[3] https://github.com/IBM/HElayers

[4] Secure Computing Interface Framework supports Industry standard Frameworks to accelerate encrypted data workloads on Cornami's FracTLcore Computing Fabric without code changes.

training algorithm for training a linear regression model on the encrypted data. For this case study, we use the gradient descent algorithm to train a linear regression model on the encrypted dataset. Once the training process is completed, the cloud server obtains the encrypted weights corresponding to the linear regression. For inference using the encrypted linear regression model, the clients first use PSI to obtain the intersection of their datasets. The clients then send the PSI result to the cloud server, which can perform the computation for inference on the encrypted data. The complete workflow of our design for training a linear regression model on encrypted data and making inference using the encrypted model is shown in Figure 1.

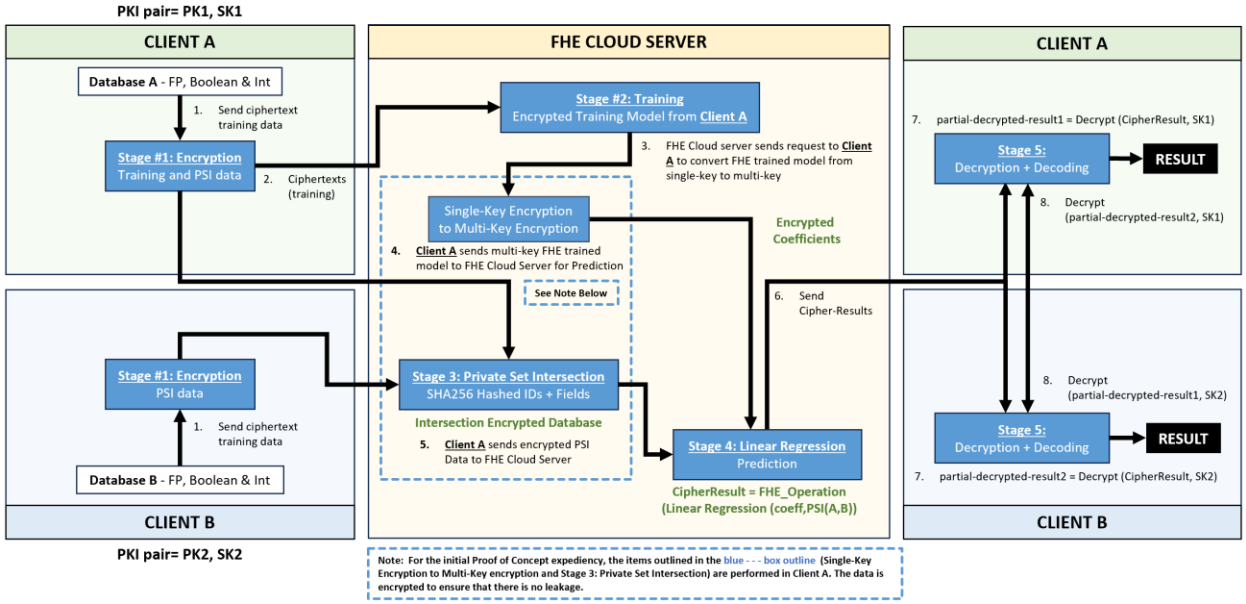


Figure 1: The Workflow for Training a Linear Regression Model on FHE-Encrypted Data and making Inference using that model.

For training the linear regression model on encrypted data we use the CKKS scheme [4]. This scheme has been implemented in various FHE libraries and frameworks, e.g., OpenFHE[5] and HEaaN[6]. To accelerate the FHE primitives and building blocks of this FHE scheme, we use Cornami hardware acceleration API's that speedup FHE operations significantly. A pictorial view of the hardware/software stack for our case study is provided in Figure 2.

[5] https://github.com/openfheorg/openfhe-development
[6] https://eprint.iacr.org/2018/1245.pdf

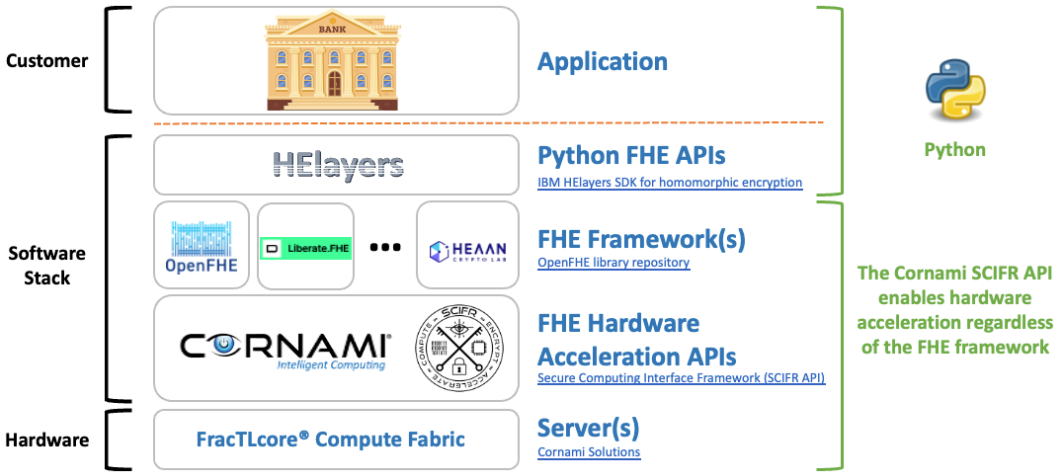


Figure 2: Architecture stack for Hardware Acceleration of FHE Primitives

3 Implementation

In this section we provide the details about our implementation of the gradient descent algorithm for training a linear regression model on a dataset.

3.1 The Gradient Descent Algorithm

For training a linear regression model on a dataset, we use the gradient descent algorithm. Given a dataset (that follows the requirements of the linear regression), we need to fit or train a linear regression model on the data. A linear regression model is shown in the following equation:

$$y = b + w_1x_1 + w_2x_2 + \cdots + w_nx_n + \varepsilon \qquad \text{Eq. 1}$$

where w_i for $i = 1, 2, \dots, n$ are the weights or coefficients of the model; and b is the intercept. Initially, the weights of the model are unknown and need to be calculated using the dataset.

There are different approaches for training a linear regression model on a given dataset. In this project we use the gradient descent algorithm. The gradient descent algorithm is an iterative process that uses the gradients of the cost function to find the optimal values for the **w**. Mathematically, gradient descent uses the following formula to iteratively update the weights of the linear regression model [14]:

$$w_{new} = w_{old} - \alpha\left(\frac{\partial J}{\partial w}\right) \qquad \text{Eq. 2}$$

Where $J(w, b) = \frac{1}{n}\sum_1^n(\hat{y} - y)\text{^}2$ is the cost function defined by the mean squared error. Furthermore, α is the learning rate and $\hat{y}$ is the predicted value. Therefore, the gradient descent uses the following formula to update the weights of the model until it reaches the optimum solution:

$$w_{new} = w_{old} - \frac{2\alpha}{n}(X.w - y).X^T \qquad \text{Eq. 3}$$

In the implemented solution (prototype application) we have used the equation 3.

3.2 FHE Implementation of the Gradient Descent Algorithm

For training a linear regression model on encrypted data, we need to carry out the gradient descent algorithm using the FHE primitives of an FHE scheme that can support computation on encrypted floating-point numbers. In this study, we select the CKKS FHE scheme because it supports computation on encrypted floating-point numbers. FHE schemes typically perform their operations using FHE primitives. Common high-level FHE primitives are addition and multiplication operations. However, behind the scenes, other typically compute-intensive operations are performed. Such operations include bootstrapping, key switching, rotation, as well as mathematical operations such as the number theoretic transformation (NTT) or the fast Fourier transformation (FFT).

After identifying the FHE primitives, the gradient descent algorithm can be implemented in two settings, software-only (SW-only) and hardware-accelerated (HW-accelerated). SW-only solution can be done using any open-source FHE software package. For the HW-accelerated solution, we implement the FHE primitives with HW-acceleration technologies that boost the performance of the building blocks significantly. Specifically, we consider the computational graph for the gradient descent algorithm (which is based on FHE primitives) and estimate its computational cost as well as its runtime by analyzing the cost of the utilized building blocks. We present our results in section 4. A sample computational graph for making inference using a trained linear regression model is provided in Figure 7 (section 4.4).

4 Results

In this section we describe our results in detail. One of the main parts of our study is the FHE implementation of the gradient descent algorithm, which is needed for training a linear regression model on a dataset.

Our implementation of the gradient descent algorithm transforms the algorithm to FHE primitives and operators that can be executed using FHE libraries. For this purpose, open-source FHE frameworks and software development kits (e.g., HElayers, OpenFHE, and HEaaN) can be used. HElayers is an SDK for developing efficient privacy-preserving data analytics applications on encrypted data. It can support various FHE libraries such as HElib, Microsft's SEAL, OpenFHE, HEaaN, etc. HElayers provides high-level API functions to use FHE schemes for developing FHE-based applications. Specifically, we use HElayer's API functions in Python to use the CKKS FHE scheme for training a linear regression model on FHE-encrypted data. We refer to this approach as software-only (or SW-only) implementation. In addition, the compute-intensive FHE operations can also be executed using hardware-acceleration technologies. We refer to this approach as the

hardware-accelerated implementation (HW-accelerated for short). In the next section, we provide our results.

4.1 SW-Only Performance Results

The gradient descent approach allows us to train a linear regression model on a variety of datasets. In particular, we focus on datasets that can have up to 50 columns for independent variables (features) and up to 100,000 rows. To measure the software-only (SW-only) performance results, we selected various datasets with different numbers of rows and columns. We ran several experiments by fixing the number of columns to five, thirty and fifty. For each case, we used various number of rows and measured the run time for training a linear regression model and making prediction with the trained model. The results are shown in the following tables and plots.

Table 1 provides a summary of the dataset dimension and the training runtimes in plaintext and FHE implementations using datasets with 50 columns and varying number of rows to train the linear regression model. Figure 3 shows a visualization of how the runtime increases as we increase the number of rows in the training dataset. Accordingly, Table 2 provides the runtime for making predictions and a summary of the dataset that was used for making prediction. Figure 4 shows a visualization of the runtime for making prediction with FHE-trained model as well as with the plaintext linear regression model. Figure 3 and Figure 4 shows the graphed results of the runtimes for training a linear regression model and making prediction with the trained model grows linearly.

Table 1: SW-only Runtime Results for Training (Variable Number of Rows and 50 Columns)

n_rows	FHE runtime (minutes)	Plaintext runtime (minutes)	n_cols
1000	69	0.00048	50
10000	167	0.00451	50
20000	306	0.01047	50
50000	739	0.02912	50
70000	989	0.04362	50
100000	1485	0.06147	50

Table 2: SW-only Runtime Results for Prediction (Variable Number of Rows and 50 Columns)

n_rows	FHE runtime (minutes)	Plaintext runtime (minutes)	n_cols
250	3.33	5.68333E-05	50
2500	28.6	0.000058	50
5000	66.2	0.00006	50
12500	138.11	8.43333E-05	50
17500	205.28	8.78333E-05	50
25000	331.28	9.18333E-05	50

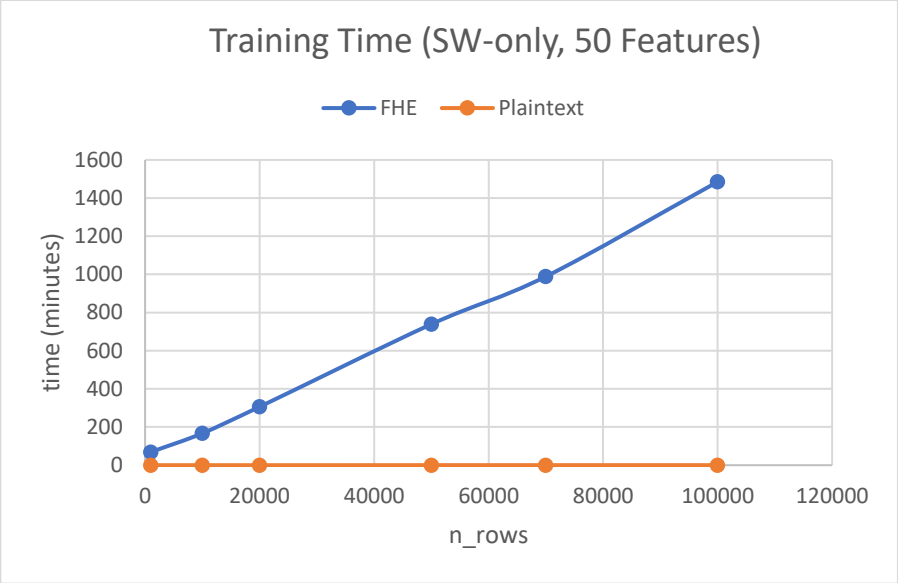


Figure 3: SW-only Training Runtime for Datasets with Different # of Rows and 50 Columns

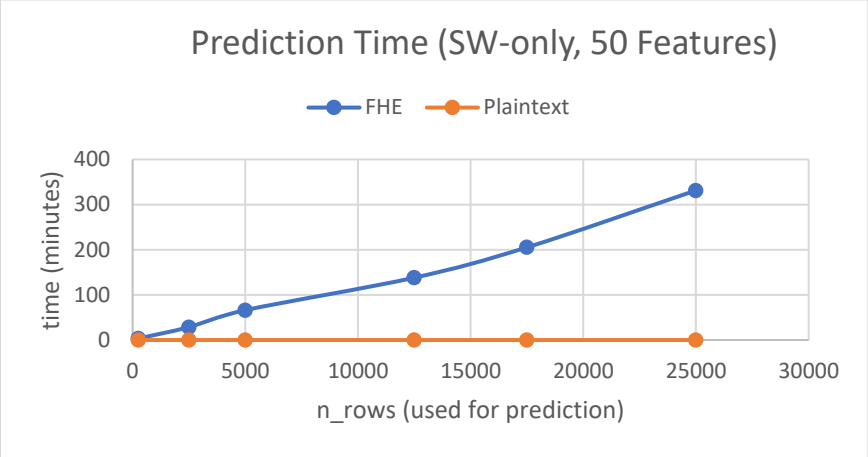


Figure 4: SW-only Prediction Runtime for Datasets with Different # of Rows and 50 Columns

4.2 HW-Accelerated Performance Results vs SW-Only Results

We carried out various experiments to measure the performance of the implemented solution for training linear regression model. We conducted two sets of experiments, namely SW-only and SW+HW configurations. The SW-only configuration is used in a test environment in which we use FHE software and packages to measure the performance of the FHE implementation of the gradient descent algorithm. The SW+HW configuration is used for testing the prototype application where the compute-intensive operations are carried out in a hardware emulation test environment. For both test configurations, the input data is encrypted using the CKKS FHE scheme. Our results for encrypted SW and SW+HW configuration are provided in Figure 5 and Figure 6.

The performance results are for the two main stages of the linear regression model, namely training the model and inference (making prediction) using the trained model. For the training stage of the project, we used the gradient descent approach. For the prediction stage, we used the encrypted coefficient that were obtained from the training stage. The SW-only experiments were performed in a bare-metal AWS instance with an AMD EPYC 7R32 2.8 GHz CPU processor with 96 cores. The HW-accelerated performance results are obtained by estimating the performance of the FHE primitives on a hardware emulator environment.

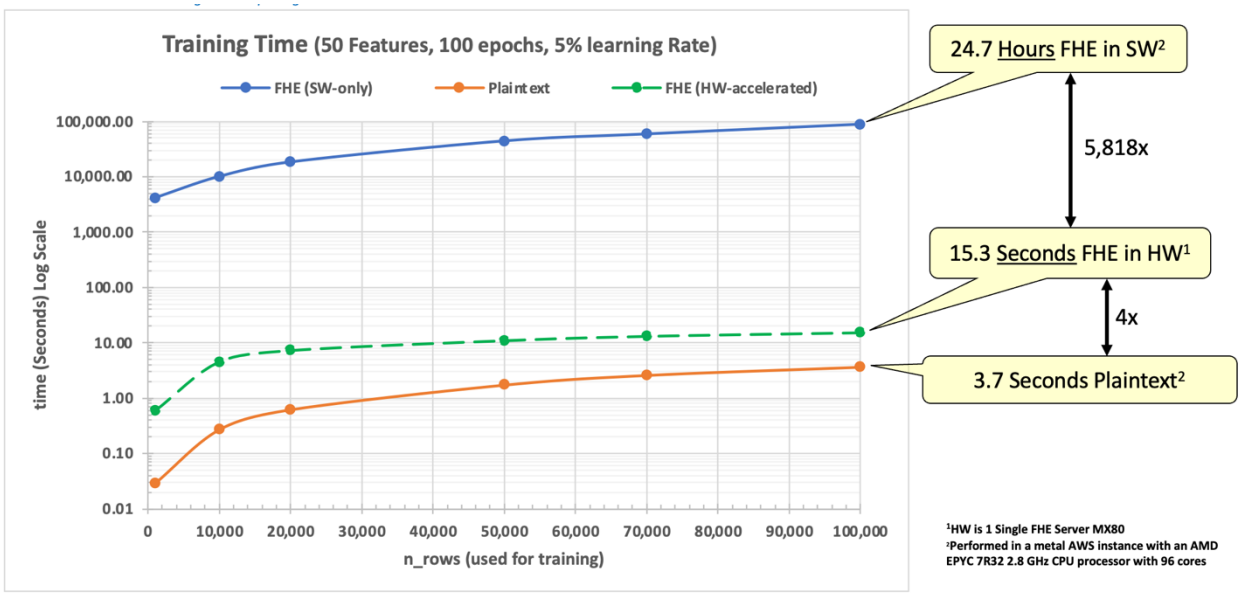


Figure 5: Training Runtime for Plaintext, FHE SW-only, Cornami FHE HW Configurations

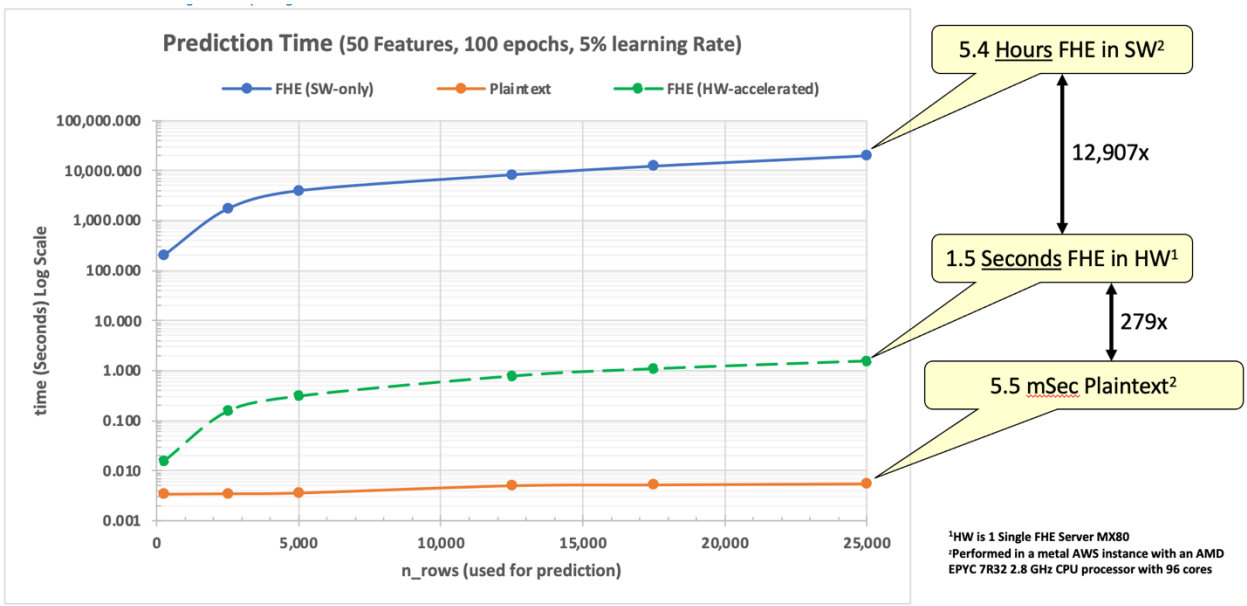


Figure 6: Prediction Runtime for Plaintext, FHE SW-only, Cornami FHE HW Configurations

Figure 5 shows the runtime for training a linear regression model in three different configurations, namely plaintext computations, SW-only computations, and HW-accelerated computations. Similarly, Figure 6 shows the runtime for prediction using the trained linear regression model in the three aforementioned settings.

Figure 5 and Figure 6 show that the HW-accelerated implementation offers a significant speedup compared to SW-only implementation. *The HW-acceleration implementation can speed up the training up to **5818 times*** (*which corresponds to reducing the training time **from 24.7 hours to 15.3 seconds***). Regarding the prediction phase, the *HW-accelerated solution can speed up the computation **12907 times** (that results in reducing the prediction time **from 5.4 hours to 1.5 seconds**).* Furthermore, as the plots show, the training and prediction time follow a linear pattern. This means that the implemented solution scales – the same improvement in performance can be realized for larger datasets.

4.3 Analysis of Model Coefficients and Relative Errors

The tables below provide a comparative analysis of model coefficients obtained using three different regression approaches—LinearRegression, SGDRegressor, and FHE Regression—across datasets of varying sizes (300, 600, 6000, and 30,000 observations). Table 3 presents the estimated intercept and coefficients for each model, while Table 4 quantifies the relative error of FHE coefficients compared to the plaintext models (LinearRegression and SGDRegressor).

From the data, we observe that as the number of observations increases, the FHE coefficients progressively approach those of the plaintext models. This trend is particularly evident in the decreasing relative errors, which indicate an improved approximation of traditional regression results by the FHE model as the dataset size grows. Notably, for 30,000 observations, the relative errors of FHE compared to LinearRegression are below 2% for both coefficients, and the differences with SGDRegressor are similarly small.

These findings suggest that while FHE-based regression initially exhibits noticeable discrepancies, its performance converges towards that of standard methods with larger datasets. This supports the potential viability of privacy-preserving regression techniques in practical applications, provided that sufficient data is available for training.

Table 3: Comparative Analysis of Model Coefficients using Three Different Regression Approaches

Model	Observations	Intercept	Coeff. 1	Coeff. 2
Sklearn LinearRegression	300	0.0000	0.8657	-0.0137
Sklearn LinearRegression	600	0.0000	0.8037	0.4713
Sklearn LinearRegression	6000	0.0000	0.7480	0.0782
Sklearn LinearRegression	30000	0.0000	0.4815	0.8200
SGDRegressor	300	0.0009	0.8589	-0.0108
SGDRegressor	600	-0.0010	0.8039	0.4698
SGDRegressor	6000	0.0009	0.7441	0.0701
SGDRegressor	30000	-0.0012	0.4805	0.8193
FHE Regression	300	0.0000	0.8510	-0.0123
FHE Regression	600	0.0000	0.7910	0.4650
FHE Regression	6000	0.0000	0.7354	0.0772
FHE Regression	30000	0.0000	0.4732	0.8061

Table 4: FHE Regression Across Datasets of Varying Sizes

Observations	Relative Error (FHE vs LinearRegression) - Intercept	Relative Error (FHE vs LinearRegression) - Coeff. 1	Relative Error (FHE vs LinearRegression) - Coeff. 2	Relative Error (FHE vs SGDRegressor) - Intercept	Relative Error (FHE vs SGDRegressor) - Coeff. 1	Relative Error (FHE vs SGDRegressor) - Coeff. 2
300	3.01e-07	1.11%	18.88%	1.59e-05	0.87%	14.81%
600	2.96e-07	1.57%	1.34%	1.56e-05	1.61%	1.10%
6000	3.92e-08	1.02%	3.87%	1.00e-06	1.17%	10.04%
30000	9.64e-09	1.72%	1.70%	2.28e-06	1.52%	1.60%

4.4 FHE Hardware Performance

The FHE hardware performance results are obtained based on the implementation of the gradient descent using FHE primitives. We use HElayer's API functions, in Python, to implement the gradient descent based on the FHE primitives of the CKKS scheme.

The Linear Regression Python code is processed by HElayers and produces a dataflow graph consisting of FHE primitive operations and their interconnections. The graph on Figure 7 is for inference or prediction on a dataset with an arbitrary number of rows (1 to unlimited) by 50 columns. The fifty FHE encrypted data points (either coefficients or input values) are packed (SIMD) across 7 FHE encrypted input vectors. The encrypted data processing flow is described below:

1. The graph reads in the encrypted coefficients (7 tensors).

2. The graph then reads in the 1st row of encrypted input data (7 tensors).
3. An encrypted matrix multiplication is performed, and the encrypted results across the 7 tensors are summed together (sum of tree) to produce the single encrypted predicted output value.
4. The encrypted predicted tensor is then sent to the output.
5. GOTO step 2 to continue processing until all the input rows are processed.

To estimate the performance, we collected FHE primitives from the hardware RTL emulator. Each FHE primitive has size (number of FracTLcores), latency (clock cycles), and throughput (Clock cycles per Output) values. For the given number of FracTLcores in a server, the graph is processed (summation of the processing chain of FHE primitives) to determine the total processing time for the total number of inputs.

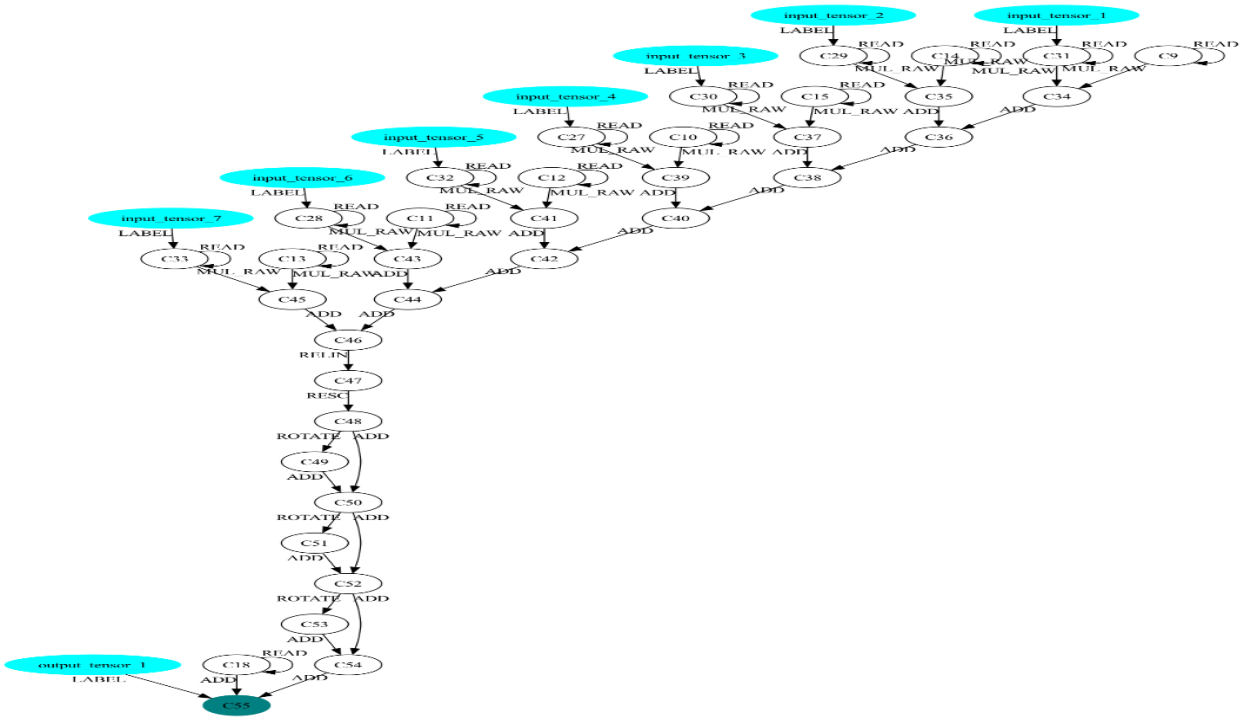


Figure 7: Call graph for Regression inference on HW Emulator

4.5 Scaling the FHE Application

We implemented a prototype application for training a linear regression model on FHE-encrypted data and for making inference on FHE-encrypted data. The implemented solution implements the gradient descent approach for training a linear regression model on encrypted data. The test results show that the runtime for training a linear regression model grows linearly (see Figure 3). Similarly, the runtime for making inference using a trained linear regression model grows linearly (see Figure 4). The obtained test results and observations show that the implemented solution is

scalable and can support large-scale data analytics applications, that use datasets with millions of rows. This observation can be seen in Figure 5 and Figure 6, for the training and prediction phases respectively.

5 Conclusion

FHE offers promising solutions for ensuring data security and privacy. Unlike conventional cryptography that only provides data security in storage and/or transmission, FHE solutions also enable performing computations on encrypted data (data security for data in use). There has been a significant amount of progress on theory and development of FHE schemes. To effectively integrate the FHE-based solutions into real-world applications, we need to speed up the FHE schemes with hardware acceleration technologies.

In this paper, we used one of the popular FHE schemes, the CKKS scheme, for a real-world multiparty computation data analytics application. We implemented the gradient descent algorithm to train a linear regression model on an FHE encrypted dataset. We used the trained linear regression model to make predictions on the private set intersection (PSI) of two datasets. Our results from this project show that the HW-accelerated implementation of the gradient descent algorithm provides a significant speedup for training and prediction. HW-accelerated FHE solutions enables the use of FHE in real-world applications.

6 References

[1] R. L. Rivest, M. L. Dertouzos, and L. Adleman, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, 1978.

[2] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2009. doi: 10.1145/1536414.1536440.

[3] M. Albrecht *et al.*, "Homomorphic encryption standard," in *Protecting Privacy through Homomorphic Encryption*, 2022. doi: 10.1007/978-3-030-77287-1_2.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017. doi: 10.1007/978-3-319-70694-8_15.

[5] B. Chen and X. Zheng, "Implementing Linear Regression with Homomorphic Encryption," *Procedia Comput Sci*, vol. 202, pp. 324–329, 2022, doi: 10.1016/j.procs.2022.04.044.

[6] P. M. Esperança, L. J. M. Aslett, and C. C. Holmes, "Encrypted accelerated least squares regression," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, 2017.

[7] W. J. Lu, S. Kawasaki, and J. Sakuma, “Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data,” in *24th Annual Network and Distributed System Security Symposium, NDSS 2017*, 2017. doi: 10.14722/ndss.2017.23119.

[8] D. Wu and J. Haven, “Using homomorphic encryption for large scale statistical analysis,” *FHE-SI-Report, Univ. Stanford, Tech. Rep. TR-dwu4*, 2012.

[9] C. Guo and N. J. Higham, “A Schur–Newton Method for the Matrix \lowercase{\boldmath *p* }th Root and its Inverse,” *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 3, pp. 788–804, Jan. 2006, doi: 10.1137/050643374.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(Leveled) fully homomorphic encryption without bootstrapping,” in *ITCS 2012 - Innovations in Theoretical Computer Science Conference*, 2012. doi: 10.1145/2090236.2090262.

[11] J. Fan and F. Vercauteren, “Somewhat Practical Fully Homomorphic Encryption,” *Proceedings of the 15th international conference on Practice and Theory in Public Key Cryptography*, 2012.

[12] R. Hall, S. E. Fienberg, and Y. Nardi, “Secure multiple linear regression based on homomorphic encryption,” *J Off Stat*, vol. 27, no. 4, 2011.

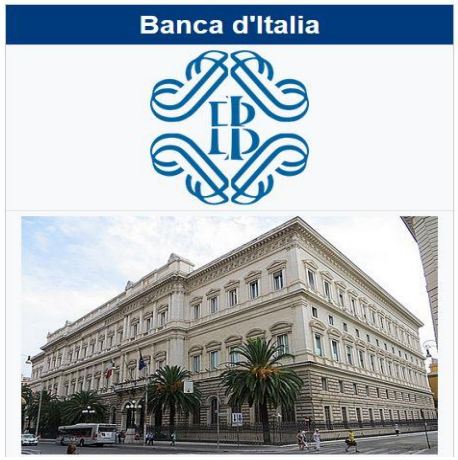
[13] A. Bertolace, K. Gatsis, and K. Margellos, “Homomorphically encrypted gradient descent algorithms for quadratic programming,” Sep. 2023.

[14] A. Ng, “CS229 Lecture notes,” *CS229 Lecture notes*, vol. 1, no. 1, 2000, doi: 10.1111/j.1466-8238.2009.00506.x.

Data sharing conundrum? In fully homomorphic encryption we (must) trust

Presenter: Adriano Baldeschi PhD

Authors: Adriano Baldeschi, Giuseppe Bruno, Mirko Avantaggiato, Andrea Capitanelli



CORNAMI®
Intelligent Computing

What is Cryptography?

- Definition:
- - Cryptography is a technique to protect information by converting it into an unreadable format without a secret key.

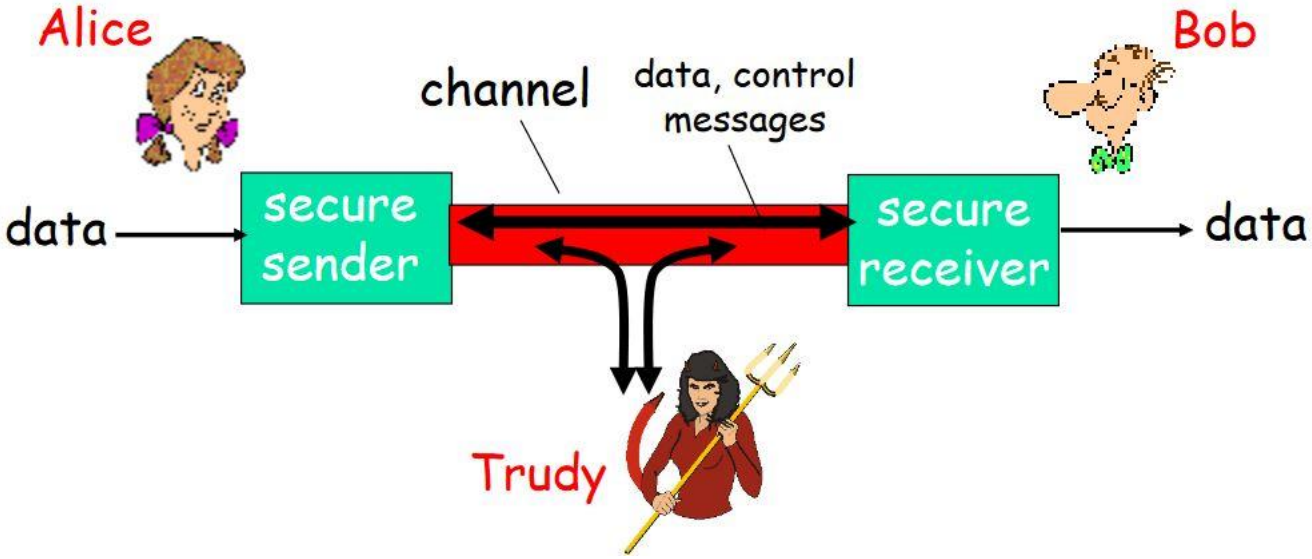
- Main Objectives:
- 1. Confidentiality
- 2. Integrity

A little bit of naming

A cryptographic system is a system capable of encrypting and decrypting a message through the use of an algorithm and a key (an alphanumeric string). The message to be encrypted is called “plaintext” while the result of the cryptographic algorithm is called “ciphertext”

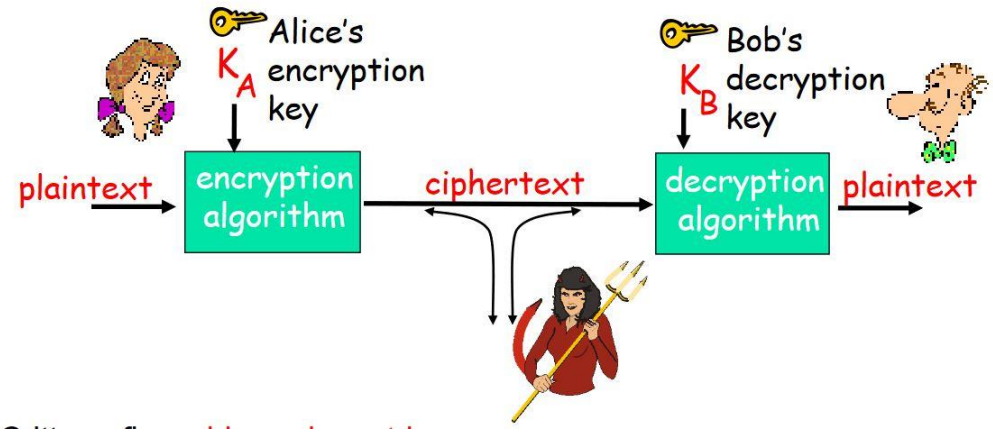
Alice and Bob are the two parties that intend to communicate securely through the network

- They could be client and server programs, or devices (e.g. routers) …
- Trudy is a third party that can listen to the messages exchanged by Alice and Bob and possibly alter them, delete them or create fake ones



Types of Cryptography

- Main Categories:
- - Symmetric Cryptography: One key for both encryption and decryption.
- - Asymmetric Cryptography: A pair of public and private keys.



Introduction to Homomorphic Encryption

- What is it?
- - A type of encryption that allows computations on encrypted data without decrypting it.

- Key Feature:
- - The computation result remains encrypted but, when decrypted, matches the result of working on plaintext data.

- Origin of the term:
- - 'Homomorphic' means 'same form' in Greek.

How Does Homomorphic Encryption Work?

- 1. Encryption: Data is transformed into encrypted form.
- 2. Computation on encrypted data: Operations like addition and multiplication are performed without access to plaintext.
- 3. Decryption: The encrypted result is transformed into plaintext, providing the correct output.

- Note: Computationally intensive but increasingly practical.

An example

Scenario: Credit Risk Analysis with Homomorphic Encryption

A bank wants to calculate a customer's **creditworthiness** based on their **income** and **total debt**, but without ever seeing the raw data for privacy reasons.

Problem:

- The customer does not want to share their income and debt in plaintext with the bank.
- The bank needs to compute the **debt-to-income ratio** to assess risk.
- If traditional encryption were used, the bank would not be able to perform calculations on the encrypted data.

Solution with Homomorphic Encryption

1. The customer **encrypts** their financial data with a homomorphic public key.
2. They send the encrypted data to the bank.
3. The bank performs computations directly on the encrypted data without decrypting it.
4. The encrypted result is returned to the customer.
5. The customer **decrypts** the result with their private key and obtains their creditworthiness score.

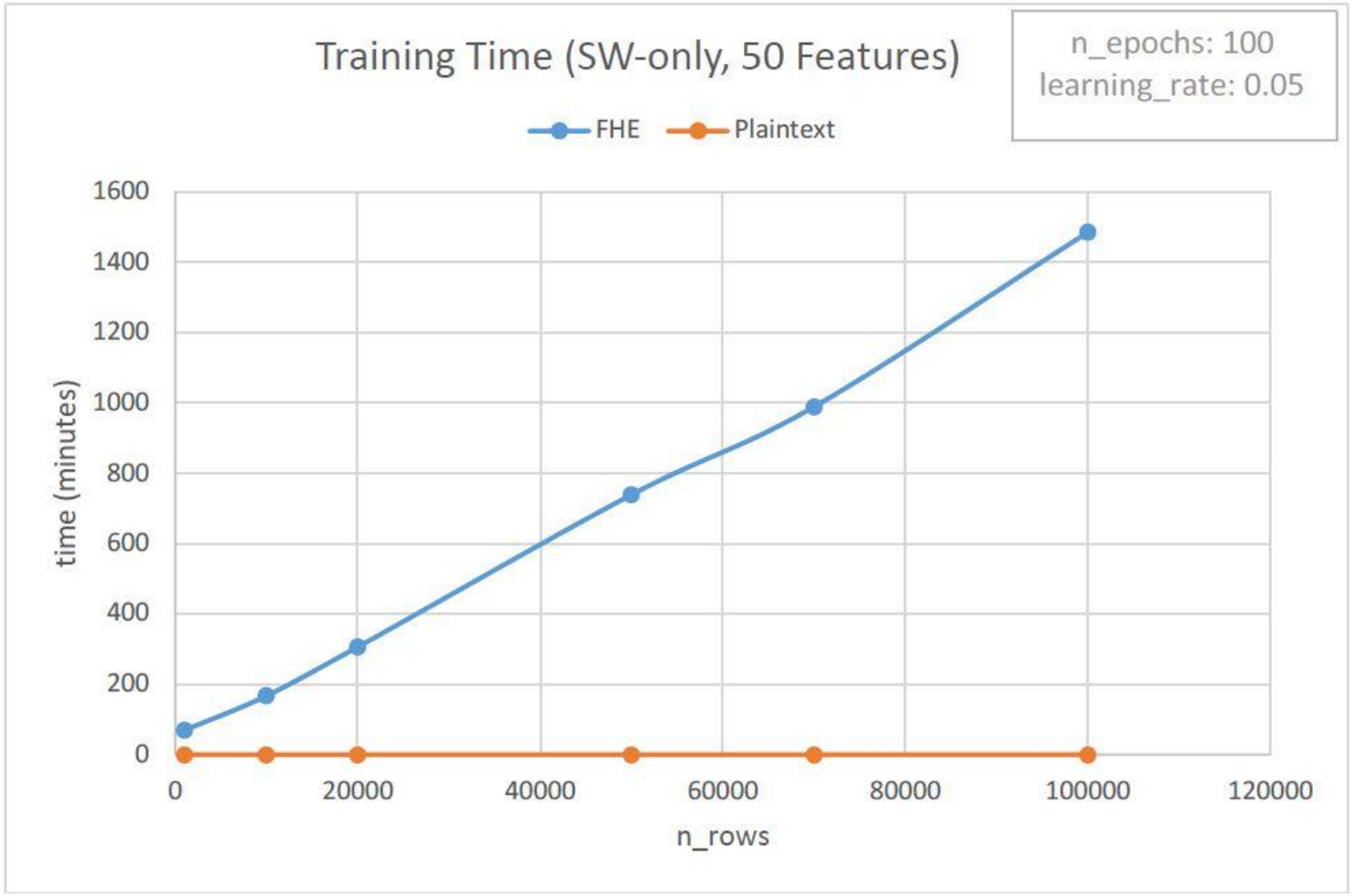
Using homomorphic encryption on linear regression

1. **Scenario**:
   - A healthcare organization wants to use patient data (e.g., age, weight, blood pressure) to train a linear regression model to predict disease risk.
   - The data is sensitive and must remain private throughout the analysis.
2. **Homomorphic Encryption in Action**:
   - The organization encrypts the dataset using a **homomorphic encryption scheme** before sharing it with a data analyst or external machine learning provider.
   - This encryption allows computations directly on the encrypted data without needing decryption.
3. **Steps in the Process**:
   - **Data Encryption**:
     Each data point (features and target values) is encrypted using a public key. For example:
     - X (features) → $Enc(X)$
     - y (target) → $Enc(y)$
   - **Secure Computation**:
     The analyst performs the required computations (e.g., matrix multiplication for gradient descent) directly on the encrypted data:
     - Compute $Enc(\beta) = (X^T X)^{-1} X^T y$, where β is the coefficient vector of the regression model.

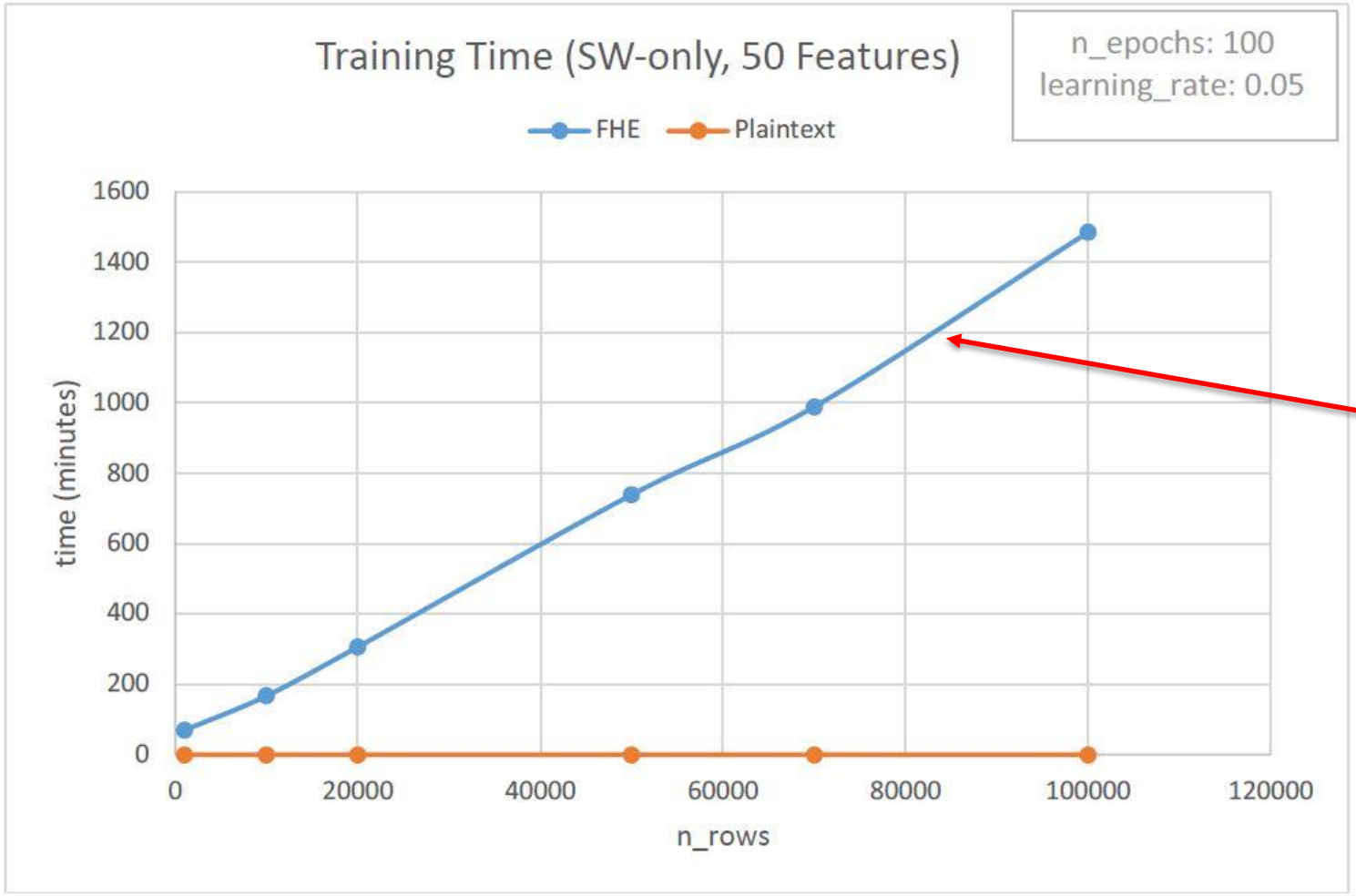
The gradient descent approach to linear regression

The gradient descent approach allows us to fit a linear regression model on a variety of datasets. In particular, we focus on datasets that can have up to 50 columns for independent variables (features) and up to 100000 rows (one hundred thousand rows). To measure the software-only (SW-only) performance results we selected various datasets with different numbers of rows and columns. Specifically, we ran several experiments by fixing the number of columns to five, thirty and fifty. For each case, we used different number of rows and measured the run time for fitting a linear regression model and making inference (prediction) with the trained model. The experimental results are as shown in the following tables and plots.

Training time



Training time



Slow!

Training time on Cornami accelerated Hardware

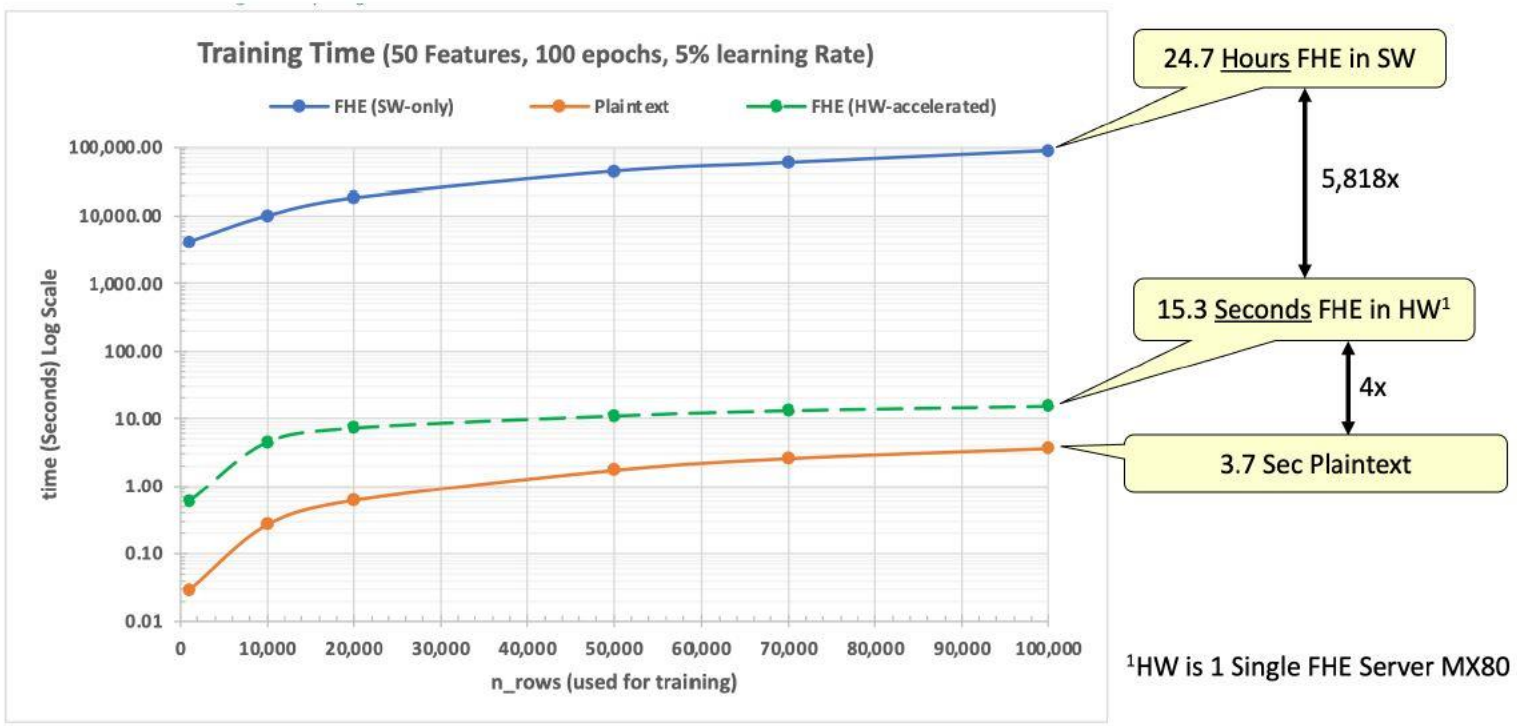


Figure 9: Training Runtime for Plaintext, FHE SW-only, Cornami FHE HW

Statistical Analysis of the Results

RMSE: root mean square error

n_rows	n_cols	RMSE (FHE and actual values)	RMSE (PTXT and actual values)
250	50	0.0064	0.00626
2500	50	0.00535	0.00499
5000	50	0.00529	0.00486
12500	50	0.0065	0.00618
17500	50	0.00683	0.00658
25000	50	0.00567	0.00533

When n_rows>1000 we obtain good results

Conclusion

- We trained a linear regression model on fully homomorphic encrypted (FHE) data using the stochastic gradient descent algorithm.
- The estimated coefficients are comparable to the actual ones when the dataset contains more than 1,000 rows.
- Although training time is relatively slow using software, it can be significantly faster using Cornami hardware.