D3.3 - Complete Specification and Implementation of Privacy Preserving Data Analytics

Work Package	WP3, Privacy Enhancing Technologies for Data Analytics
Lead Author	Sébastien CANARD, Bastien VIALLA (ORA)
Contributing Author(s)	Beyza BOZDEMIR, Orhan ERMIS, Melek ÖNEN (EURC), Allon ADIR, Ron SHMELKIN, Ramy MASALHA, Boris ROZENBERG (IBM), Sébastien CANARD, Bastien VIALLA (ORA)
Reviewers	Matthias BECKERLE (KAU), Eleonora CICERI (MCI)
Due Date	30.04.2020
Delivery	30.04.2020
Version	1
Dissemination Level	Public

The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation Programme, through the PAPAYA project, under Grant Agreement No. 786767. The content and results of this deliverable reflect the view of the consortium only. The Research Executive Agency is not responsible for any use that may be made of the information it contains.

Revision History

Revision	Date	Author	Description
0.1	08.01.2020	Bastien Vialla (ORA)	First release, table of contents.
0.2	06.04.2020	All contributing authors (ORA, EURC, IBM)	Version for the first internal review
0.3	17.04.2020	All contributing authors (ORA, EURC, IBM)	Reviewer comments fixed
0.4	21.04.2020	Bastien Vialla (ORA)	Version for the second internal review
0.5	28.04.2020	Bastien Vialla (ORA)	Reviewer comments fixed
0.6	29.04.2020	Orhan Ermis (EURC)	Quality check completed

Executive Summary

In Deliverable D3.1, we described preliminary designs for privacy preserving primitives for the three main data analytics techniques: neural network inference and training, clustering, and statistics. In this report, we present the specifications and implementations of those privacy preserving primitives together with the new privacy preserving neural network training and the new privacy preserving statistics solutions. We also present our preliminary results for these primitives. All the solutions presented in this document aim at enabling an untrusted third-party data processor to perform the underlying operations over protected data. The PAPAYA solutions are further described in four main categories:

- **Privacy preserving neural networks.** For neural networks, both the inference and the training are investigated:
  - For the privacy preserving neural network inference, we designed and implemented four solutions using different cryptographic primitives, namely two party computation (2PC), partially homomorphic encryption (PHE), fully homomorphic encryption (FHE) and a hybrid solution that combines 2PC and FHE. Performance of these solutions are compared using arrhythmia and image classification usage scenarios. Our experimental results show that there is no single technique that outperforms but each solution has some specific improvements for these usage scenarios.
  - For the privacy preserving neural network training, we present a solution based on FHE. The proposed solutions particularly investigates the problem of text classification. Our experiments show that the proposed solution provides promising results on complex networks although it has some small degradation on the accuracy.
  - Additionally, we also propose a solution based on collaborative training. We examine collaborative training with or without using differential privacy. We also investigate the effect of property inference attack on collaborative training. Our experiment results show that increasing the number of participant and decreasing the percent of the sensitive data in victims' dataset, decreases the success rate of the attack.
- **Privacy preserving clustering.** We focus on the clustering of trajectory to answer the problem of Use Case 3 (see D2.1). We consider a scenario whereby a data owner holds a dataset of trajectories and would like to delegate the trajectory clustering operations to a third-party (untrusted cloud server). A privacy preserving variant of trajectory clustering is to develop an approximated version of the original clustering algorithm and combine it with cryptographic tools in order to identify these clusters without exposing the sensitive data. We propose two solutions for privacy preserving clustering For this problem we designed two solutions:
  1. As described in deliverable D3.1, trajectory data can be analysed using a dedicated clustering algorithm named TRACLUS. Therefore, the first solution is to approximate TRACLUS algorithm and use 2PC as a cryptographic primitive.
  2. We also investigate using MinHash function as the candidate for trajectory clustering. In order to provide the privacy, we employ 2PC together with the modified version of the MinHash algorithm.

Our preliminary experiments show that both solutions can be considered as a potential solution since they able to cluster hundreds of trajectory in short time period.

- **Privacy preserving counting.** The goal of this primitive is to provide a solution for Use Case 2 (see D2.1). The primitive aims at counting people while preserving their privacy. For this purpose, we have designed a solution based on bloom filters and homomorphic encryption. Moreover, the proposed solution ensures k-anonymity. According to our experiments, our solution provides good results in terms of efficiency and also it is able to count millions of elements in less than a minute.

- **Privacy preserving statistics.** This primitive aims at providing a solution for the Use Case 4 (see D2.1). Our aim is to allow external querier to make some basic statistics queries on a data set collected from individuals without violating their privacy. For this particular data analytics, we employ functional encryption as a cryptographic primitive.

Contents

List of Figures

List of Tables

1 Introduction

One of the goals of the PAPAYA project is to investigate and design primitives for privacy preserving data analytics, and apply it to real world problems. All the solutions aim at enabling an untrusted third-party data processor to perform the underlying operations over protected data. To stay grounded in reality, partners proposed use cases representing challenges arising in practice. In Deliverable D3.1, we identified the main analytics operations needed for the use cases, namely: the training and inference of neural networks, trajectory clustering, counting and statistics. In this report we give the complete specifications, implementations and performance analysis of each primitive:

- **Neural networks.** We first present and compare different solutions designed for privacy preserving inference of neural networks Section 2. We describe four solutions for the inference of a neural network based on different cryptographic primitives in Section2.1. We then compare the performance of each solution on two different usage scenarios. Later, we show our work on privacy preserving training of neural networks using fully homomorphic encryption and its application to text classification in Section2.2. Finally, we describe our privacy preserving collaborative training solution and its potential countermeasure agains the property inference attack in Section 2.3.

- **Trajectory clustering.** For the privacy preserving trajectory clustering, in Section 3, we introduce the specifications, implementations and performance analysis of two different solutions. The first solution is the approximated version of the TRACLUS algorithm [28] that uses 2PC as a cryptographic primitive. On the other hand, the second one uses the modified version of MinHash algorithm together with 2PC as given in Section 3.2.

- **Counting.** For the privacy preserving counting, we introduce our solution in Section 4. Our solution is based on Bloom filters and homomorphic encryption that provides the privacy by ensuring k-anonymity.

- **Statistics.** Finally, we give the specifications for our privacy preserving statistics solution that uses functional encryption in Section 5.

2 Privacy Preserving Neural Networks

In this section we present our results for the privacy preserving inference and training of neural networks. Since almost all of the specifications regarding privacy preserving inference of neural networks and collaborative training introduced in D3.1, in this document, we provide an overview and preliminary results for these primitives. Furthermore, we introduce our privacy preserving training of neural networks solution in this section.

2.1 Privacy Preserving Neural Networks Inference

In this section we give an overview for our privacy preserving inference of neural networks, which are based on various cryptographic primitives such as partially homomorphic encryption (PHE), fully homomorphic encryption (FHE), secure two party computation (2PC) and hybrid solution that combines FHE and 2PC as described in D3.1.

2.1.1 Description of Case Studies

Arrhythmia classification (see Deliverable D2.1). Heart arrhythmia is a set of conditions in which the heartbeat is not regular. Most types of arrhythmia a patient can be subjected to, are not causes for concern, as they neither cause damages to the heart nor make the patient experience symptoms. Unfortunately, several arrhythmia types cause symptoms that range from tolerable ones (e.g., lightheadedness) to more serious ones (e.g., short breath), and some others predispose patients to heart failure and stroke, resulting in grave consequences such as cardiac arrests. For this reason, it becomes vital to monitor chronic patients' ECG signals to identify arrhythmias at their onset, and prevent the aggravation of patients' conditions.

Nowadays, several commercial services that perform arrhythmia detection on ECG signals can be found in the market. These services collect patients' ECG data via dedicated wearable devices, analyse them to detect arrhythmias and report the results to a healthcare professional, who creates a report. As the identification of arrhythmia in this case is done by a machine, there is a need for building reliable and accurate algorithms for the identification of critical ECG sections. In this context, the algorithmic paradigm of deep learning represents a valid tool for improving the performance of automated ECG analysis [19].

Unfortunately, there are limitations to this approach. Indeed, the burden of analyzing long streams of ECG data for a large number of patients may be difficult to be handled on premises, where the potential lack of computational resources would limit the performance. To overcome this issue, one could acquire ECG data on premises and outsource them to an external environment (with more resources), where the arrhythmia detection would be performed. Nevertheless, moving from a trusted environment to an untrusted one would endanger the protection of personal data. This aspect becomes particularly critical when analyzing health-related data, as the most recent regulations on data protection (such as the GDPR) impose strict analysis constraints for the so-called *special categories of data* (as per Article 9). Hence, it becomes essential, in this case, to protect data before outsourcing them to the untrusted environment, e.g., using advanced cryptographic techniques.

Image classification. Image classification [20] is the study of processing an image and extract valuable information from its content. Image classification has various application areas that spans from face [10] or finger print [41] recognition for biometrics to video surveillance systems [38], hand gestures recognition for sign language [8], etc.

Classifying an image involves computationally intensive operations. With the recent developments in information systems, particularly with the rise of Graphical Processing Units (GPUs), the popularity of image classification has increased again for researchers in machine learning. Thus, many small and medium organizations started developing new applications based on the purpose of image classification for either providing better services for their customers such as face recognition for access control rather than using password based access control, or surveilling an area, building, room, etc. Although GPUs provide extra computation power for the processing of an image, such companies may require the help of computationally more powerful environments such as cloud servers. Companies against face with the dilemma: outsourcing the images and the underlying image classification operations to an untrusted environment raise privacy issues since these images may contain sensitive information about individuals and, more critically, some malicious parties can gain access to various online systems using these individuals' images. Therefore, an extra layer of protection should be provided before outsourcing these images to the untrusted environment such as the use of advanced cryptographic techniques mentioned throughout the paper.

2.1.1.1 Neural Networks Architectures

In order to evaluate the suitability and efficiency of the advanced cryptographic techniques mentioned previously, we propose a comparative study for neural network classification with the two previously described use cases, namely arrhythmia and image classification. To this aim, we build a small NN model for the arrhythmia classification and a deeper NN model for the image classification. These models are newly built in order to be compatible with the use of FHE and 2PC.

For the arrhythmia classification usage scenario, the PhysioBank database[1] is employed for training and classification of the newly built NN model. An ECG beat is encoded into 180 samples and our NN model processes these samples to obtain one of the 16 arrhythmia classes as defined in [31]. The architecture of the neural network consists of 2 fully connected (FC) layers and one activation layer implementing x^2 as an activation function. The resulting model achieves 96.51% accuracy.

For the image classification usage scenario, the MNIST database[2] that consists of handwritten digits is used to construct the NN model. The architecture of the model consists of a two-dimensional convolution layer (five different 5×5 filters with a stride of $(2,2)$, followed by an activation layer that implements x^2, two FC layers and a final activation layer again implements x^2 between these FC layers. The accuracy of the model is calculated as 97.39%.

[1] `https://www.physionet.org/physiobank/database/mitdb`
[2] `http://yann.lecun.com/exdb/mnist/`

2.1.2 Privacy Preserving Neural Networks Using Two-Party Computation

This section overviews our privacy preserving neural network solution based on 2PC, which was described in deliverable D3.1. specification and description are detailed in D3.1.

We propose to use ABY [14] a framework for efficient mixed-protocol secure two-party computation that efficiently combines the use of Arithmetic shares, Boolean shares, and Yao's garbled circuits to implement and evaluate the newly designed model. ABY supports many operations for these circuits and provides novel and highly efficient conversions between different shares.

The first solution translates the Neural Network (NN) model by regrouping the operations (e.g. matrix-vector multiplication, bias addition, etc.), into Arithmetic circuits. Both the input vector and the model are represented with matrices and vectors with floating point numbers in which values are represented as doubles (64 bits variables). Each Boolean share consists of 64 wires. When working with floating point numbers, ABY builds a specific circuit for each operational gate: For example, one floating point multiplication gate consists of 3034 XOR gates, 11065 AND gates and 3 MUX gates. Consequently, the total number of gates in the resulting circuit becomes 669,854 and the depth of the circuit is evaluated as 5,330.

The multiplication and addition of Boolean shares are much more time and bandwidth consuming than multiplication and addition of arithmetic shares. We, therefore, consider the use of arithmetic circuits only and represent real numbers with fixed-point numbers. As the multiplication of two fixed-point numbers can yield numbers with a number of bits higher than the two initial numbers, hence to an overflow, these numbers need to be truncated and/or rounded in order to ensure that all intermediate values can be represented in 64 bits.

2.1.3 Specifications and Implementation

We propose to use the ABY framework [15] to realize additions and multiplications of the proposed neural networks models described in subsection 2.1.1. In particular, we propose to use arithmetic circuits in the ABY framework since the majority of the underlying operations are linear (matrix multiplications) and there are no comparisons. Moreover, we approximate all real numbers into integers by using a simple truncation method that consists of keeping only some digits of the fractional part (hence by multiplying them with 10^n). The resulting circuit for privacy preserving arrhythmia classifier has depth 5 and 127 arithmetic gates and for privacy preserving image classifier, the circuit has depth 7 and 37685 arithmetic gates. Both classifiers also allow for prediction in batches thanks to the use of the SIMD packing method. See deliverable D3.1 for more details. The performance results for the proposed solutions are given in Subsection 2.1.7

2.1.4 Privacy Preserving Neural Networks Using Partially Homomorphic Encryption

Another alternative to develop a privacy preserving neural network classification scheme is to use partially homomorphic encryption. As introduced in deliverable D3.1, we propose a solution, named SwaNN [40] that ensures privacy through the use of additively homomorphic encryption (AHE) with two-party computation (2PC). Similarly to [17], the solution consists of approximating the activation layer to the square function and involving the client on this computation because AHE cannot support the multiplication of encrypted data. The client is also involved on the max pooling layer through 2PC. The specification of the client-server solution is described in D3.1. In this deliverable, we describe a server-aided setting whereby the entire set of operations is delegated to two non-colluding cloud servers. This way, the client does not participate into the classification, at all. It only sends the query and receives the classification result.

2.1.4.1 Goal

The ultimate goal remains the same as for all existing solutions (including the ones described in the previous section): to provide privacy regarding both the classification query and its results. Since the previously described solution involves the client during the actual classification, we aim at reducing the computational cost and delegate all the operations (including the square and max computation) to the server. Additionally, we also propose to minimize the amount of computations at the client side and the overall computational cost for the proposed model.

2.1.4.2 Specifications and implementation

As previously introduced, we design a scenario which enables the client to outsource the computations to two non-colluding servers. In this scenario, the client provides the input to one server who performs all the linear operations and once a layer with non-linear operations is reached, the two servers perform the operations, jointly. Furthermore, since the other server remains idle until all the linear operations are performed, we propose to provide one different encrypted image to each server to fully utilize the computation capabilities of the servers and classify two images at once. Figure 1 illustrates our scenario.

Description. The client creates shares of the private key for each server as described in [13] and sends the shares to each server. We divide the computations into two phases: In the non-interactive phase, the servers compute the linear operations on their inputs as the same way described in deliverable D3.1; The interactive phase is also similar to the one described for the client-server setting, except in the decryption procedure. Indeed, the decryption task is also delegated to the servers along who have initially received their shares on the secret key. To clarify the procedure, in Protocol 1 we illustrate how secure square protocol works when the computations are delegated to the two servers.

Optimizing computations. We propose several optimization techniques which help reduce the computation and communication cost. The first one is to use an efficient packing technique while processing the inputs. With this technique, instead of processing multiple inputs separately, they are regrouped into a single input and processed at the same time as described in [5]. Therefore, in our solution, we create slots of $t + \kappa$ bits for each data item where κ is the security parameter and t is the length of the data item. Given the plaintext modulus N, we can

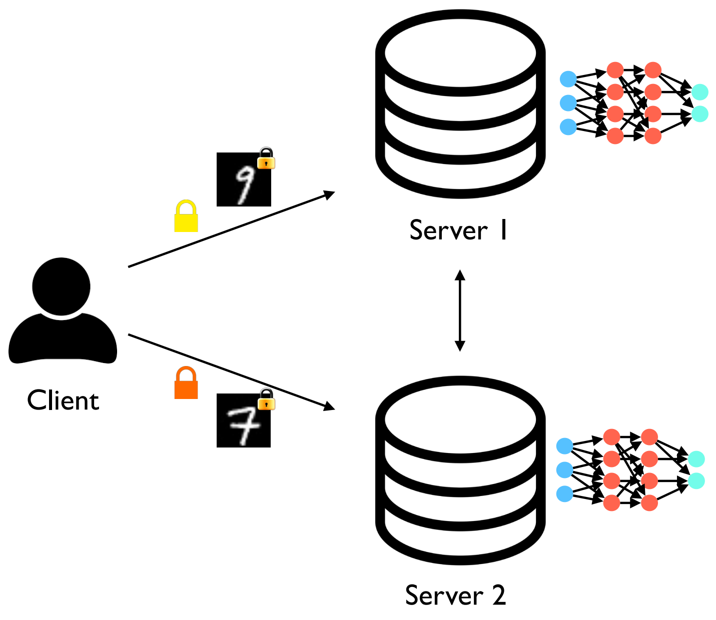

Figure 1: Two-server scenario for SwaNN with two input images.

pack $\rho = \left\lfloor \frac{\log_2 N}{t+\kappa} \right\rfloor$ items in a single ciphertext as in (1).

$$[\hat{x}] = \sum_{m=0}^{\rho-1} [x_{i,j}] \cdot (2^{t+\kappa})^m \tag{1}$$

Since our solution is based on Paillier cryptosystem [34], we are also able to use data packing for additions on the packed ciphertext, which introduces a significant performance gain.

Multi-exponentiation algorithm [29] is used as a second optimization technique in our solution

Protocol 1: Secure Square Protocol in the two-server scenario.

Server 1 (pk, sk_1)		Server 2 (pk, sk_2)
		$[x], r \in_R \{0,1\}^{\ell+\kappa}$
		$[x_r] \leftarrow [x] \cdot [r]$
		$[x_r] \leftarrow [x+r]$
$x_r \leftarrow$ decr$_1([x_r]')$	$\xleftarrow{[x_r]'}$	$[x_r]' \leftarrow$ decr$_2([x_r])$
$x_r^2 \leftarrow x_r \cdot x_r$		
$[x_r^2] \leftarrow$ enc(x_r^2)	$\xrightarrow{[x_r^2]}$	$[x_r^2] \cdot \left([r^2] \cdot [x]^{2r}\right)^{-1}$
		$[x^2] \leftarrow [x_r^2 - r^2 - 2xr]$

while simultaneously performing the operations in the form of

$$\prod_{i=1}^{w} a_i^{b_i} = a_1^{b_1} \cdot a_2^{b_2} \dots a_w^{b_w}. \tag{2}$$

With this technique, it is possible to reduce the cost of dot products and matrix multiplications for convolution and fully connected layers.

2.1.4.3 Performance

We have evaluated the performance of Swann in the client-server and the two-server settings with the x^2 activation function. For each scenario, we designed two different cryptographic settings. The first setting is an **only-PHE** setting which is totally based on the Paillier cryptosystem. We implemented the activation function x^2 as described in Protocol 1. The second setting is a **hybrid setting** where the computations realized by using PHE and 2PC.

Table 1 shows the performance results of SwaNN for both scenarios in the only-PHE and the hybrid setting for each layer of the network. For the only-PHE setting we provide the timings with and without optimizations. For the hybrid setting, we provide only optimized timing values.

Table 1: Computation time (in ms) for Convolution (`Conv`), Activation (`Act`), Pooling (`Pool`) and Fully-Connected (`FC`) layers in the client-server and the two-server scenario. The timings are provided for optimized and non-optimized PHE-only setting and optimized hybrid setting. The total timings marked with * show the simultaneous run time of SwaNN for two images.

	Non-optimized - PHE only				Optimized - PHE only				Optimized - Hybrid			
Layer	Client	Server	Server-1	Server-2	Client	Server	Server-1	Server-2	Client	Server	Server-1	Server-2
`Conv`	–	1831	1883	1883	–	892	917	911	–	917	919	900
`Act`	12651	15805	33442	33319	2487	19253	23973	23941	2292	566	2947	2984
`Pool`	–	35	34	34	–	34	34	33	–	33	33	34
`Conv`	–	2799	2911	2948	–	1329	1347	1344	–	1386	1378	1364
`Pool`	–	37	37	37	–	38	38	37	–	37	37	39
`FC`	–	6420	6579	6536	–	3809	3802	3818	–	3989	3973	3977
`Act`	1504	1879	3993	4009	314	2231	2795	2797	273	266	607	573
`FC`	–	10	10	10	–	11	11	10	–	11	11	11
Total	42972		48892*		30399		32902*		9841		9904*	

The results show that in the client-server scenario when no optimizations are used, the prediction of one image is computed approximately in 43 seconds. However, when we use optimization techniques, we can reduce the computation time to 30 seconds. In a hybrid setting, this cost is reduced to 10 seconds. Furthermore, in the two-server scenario with a slight increase in computation time, two images can be processed simultaneously. More particularly in

Table 2: Detailed computation time for the activation layer in the client-server and the two-server scenario for the hybrid setting (in ms).

Operation	Client	Server	Server-1	Server-2
Packing	–	409	413	406
Decryption	72	–	147	146
Unpacking	0.1	–	0.1	0.1
ABY	11	14	28	28
Encryption	2220	158	2373	2685
Total	2884		3265*	

an optimized hybrid setting the two servers can compute the prediction result for two images in 10 seconds simultaneously.

In Table 2, we provide the details of the computation time for the activation layer in the hybrid setting. The packing, decryption and unpacking operations are performed during the switching from PHE to 2PC. The encryptions are computed by both parties when switching the operations from 2PC to PHE. In the client-server scenario, the client spends 2.3 seconds for the computations while the server spends approximately 581 milliseconds. In the two-server scenario, each server takes approximately 3.2 seconds (3265 corresponds to the maximum total time between S1 and S2) which corresponds to the classification of two images.

Apart from computation time, we also analyzed the bandwidth usage of SwaNN for different settings. Table 3 shows the communication cost in both scenarios for the only-PHE setting and the hybrid setting. The packing technique used in the activation layers helps reduce the bandwidth usage by half. Besides due to the interactive nature of 2PC, the bandwidth usage in the hybrid setting is higher than the only-PHE setting for both scenarios.

Table 3: Bandwidth usage of SwaNN in different settings (in MB).

	Client-Server	Two-Server
PHE only (w/o opt.)	0.97	0.96
PHE only (w/ opt.)	0.51	0.51
Hybrid (w/ opt.)	1.69	1.69

As a final analysis, in Table 4 we compare SwaNN with the state-of-the-art works CryptoNets [17] and MiniONN [30] with respect to computation time and bandwidth usage. The performance results of CryptoNets and MiniONN are taken from the respective papers. According to [17] CryptoNets, which uses fully homomorphic encryption for computations, requires 297.5

Table 4: Comparison with the state-of-the-art in exp. 1.

	Computation time (s)	Bandwidth usage (MB)
CryptoNets [17]	297.5	372.2
MiniONN [30]	1.28	47.6
SwaNN	9.9	1.69

seconds for one prediction. The protocol enables simultaneous computation by packing 4096 images into a single ciphertext. This is an advantage when the same client has very large number of prediction requests. MiniONN can compute the prediction result for the same network in 1.28 seconds ([30]). However, this computation requires 47.6 MB bandwidth usage. SwaNN can compute the same prediction result in 10 seconds. Although the computation time of SwaNN is higher than MiniONN, SwaNN achieves a 28-fold less bandwidth usage.

To summarize, we believe that SwaNN actually achieves the best of both worlds, namely, better computational overhead compared to HE-based solutions and, better communication overhead compared to 2PC-based solutions. Thanks to the use of the Paillier encryption algorithm for linear operations with some optimizations based on data packing and mutli-exponentiation the solution achieves better computational cost compared to existing HE-based solutions. The communication cost is also minimized since 2PC is only used for non-linear operations (max pooling). Finally, SwaNN can be executed in the two-server setting, in case the client lacks resources.

2.1.5 Privacy Preserving Neural Networks Using Homomorphic Encryption

In this section we explain our solution for the the privacy preserving inference of a neural networks using homomorphic encryption.

The goal of the solution is the same as for the other solutions proposed: providing privacy for the classification of sensible data. There are two major benefits for using homomorphic encryption in this context:

- only two communications between the server and the client are needed during the operation. First, when the client sends its data, and second, when the client receives the encrypted result. Unlike others solutions, there is no communications during the classification per se.
- Homomorphic encryption scheme offer an efficient packing method that allows to assemble many inputs into a single ciphertext. Using the packing allows to reduce the number of ciphertexts sent to the server while processing large amount of data.

Before going into the specifications, let us recall that homomorphic encryption is a noise based encryption. During the encryption, a small noise is added to the plaintext, and as the computations go along the noise grows into the ciphertext. If the noise grow too much, it is

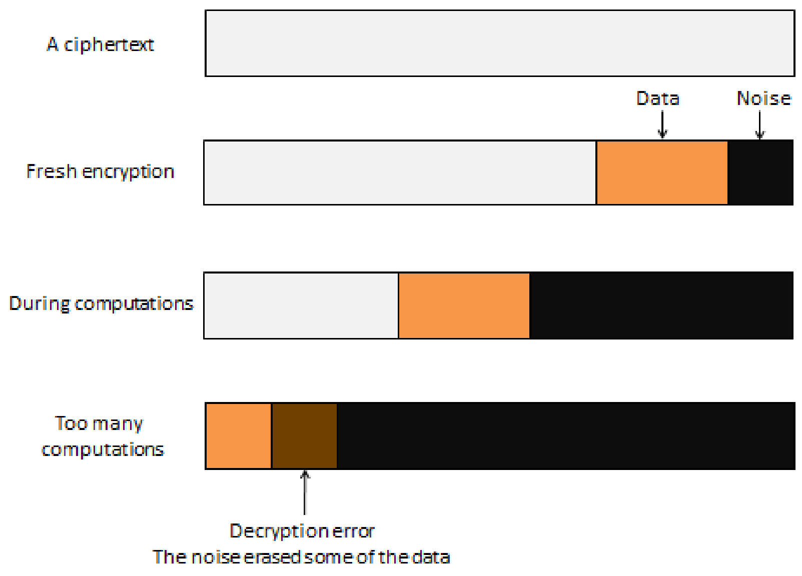

Figure 2: Noise growth in a ciphertext.

impossible to recover correctly the plain message during the decryption, see Figure 2. The more computations are done, the more room is needed in a ciphertext. The choice of scheme's parameters is dependent on the computations that will be done homomorphically. Therefore, contrary to the others primitives described in this report we cannot give general specifications for the use of homomorphic encryption for privacy preserving inference. The parameters of the encryption scheme have to be tailored for each use-case individually.

In the next subsection, we present how we applied homomorphic encryption for the use-case of arrhythmia classification.

2.1.5.1 Specifications of arrhythmia classification using homomorphic encryption

The context of arrhythmia classification is given in subsection 2.1.1.

The CKKS scheme. For this problem we use the CKKS scheme implemented in Microsoft SEAL Library [3]. We chose this scheme because it can handle floating point numbers natively.

In CKKS a ciphertext is composed of 2 polynomials of degree m with coefficient modulo a number $q = \prod_i^n p_i$, its memory footprint is $O(m \cdot 64 \cdot n)$ bits. The parameter q represent the room available for the noise to grow see Figure 2, the p_i are primes and $\{p_i\}_{i \in [0,n]}$ is called the modulus chain. The primes p_i are specials, they need to have the property that the finite field $\mathbb{F}_{p_i}$ contains $2m$ primitive root of unity. This property is needed because the SEAL library uses Fast Fourier Transform (FFT) for the multiplication of two ciphertexts. The complexity of the arithmetic operations involving CKKS ciphertexts is $O(n \cdot m \log m)$, where n is the number

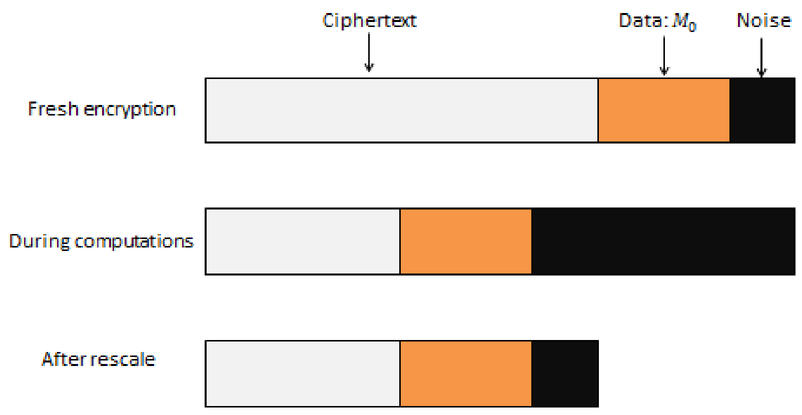

Figure 3: Effect of rescaling on the noise level of a ciphertext.

of prime in the modulus chain. To obtain the best performance we need to keep n and m as low as possible.

Let c_0, c_1 be two ciphertexts, with their noise level n_{c_0}, n_{c_1} respectively. For the arithmetic operations the noise growth is as follow:

HEAdd for $c_{\text{add}} = c_0 + c_1$, the noise level is $n_{c_{\text{add}}} = \max(c_0, c_1) + 1$.

HEMult for $c_{\text{mult}} = c_0 \cdot c_1$, the noise level is $n_{c_{\text{mult}}} = 2\max(c_0, c_1)$.

The noise growth is the same for multiplying a ciphertext by a scalar.

To control the noise growth, CKKS as a procedure called **HERescale** which drops a prime from the prime chain, hence changing q during the computation. As we can see in Figure 3 after the rescale the noise level in a ciphertext has dropped allowing to proceeds further with the computation. If we hadn't applied the rescale, we wouldn't have enough room in the ciphertext to allow more computations.

Finally, the noise level after the encryption is the same as the chosen floating point precision. For example, for the arrhythmia classification we chose to have a floating point precision of 20 bits, so a fresh ciphertext have 20 bits of noise.

Homomorphic version of the neural network. We choose a floating point precision of 20 bits for this use-case. So a fresh ciphertext have 20 bit of noise. In the CKKS scheme description we pointed out that the noise level doubles after each multiplication, which mean that the noise level will grows exponentially with the number of layers in the network. To manage the growth we can rescale the ciphertext. For each rescale we need a new prime in the modulus chain, increasing the cost in memory and computation. We have two obvious strategies:

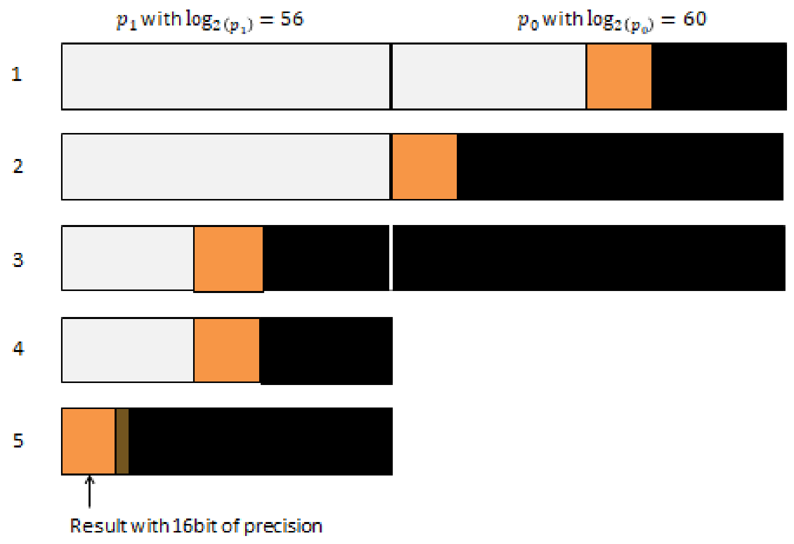
p_1 with $\log_{2(p_1)} = 56$
p_0 with $\log_{2(p_0)} = 60$
1
2
3
4
5
Result with 16bit of precision

Figure 4: Evolution of the noise level of a ciphertext during the inference of the neural network.

1. no rescale leads to q of at least 180 bits;
2. we rescale after each layer leads to 4 primes in the modulus chain.

Both strategies give large memory footprint and non optimal performance.

Our strategy is to choose 2 primes $q = p_0 \cdot p_1$, with $\log p_1 = 56$ and $\log p_0 = 60$, and only one rescale after the activation layer. With q of 116 bits we can deduce from the standard $m = 4096$. The noise analysis can be seen in Figure 4:

1. a fresh ciphertext, in orange the message and in black the noise.
2. The noise level after the first linear layer. We can see that if we apply the rescale now, we would drop all the information.
3. The noise level after the activation layer. We can go further without applying a rescale.
4. The noise level after the rescale.
5. The noise level after the last linear layer. We see that we loose 4 bits of floating point precision, with only 16 bits for the result.

We decide p_1 to be of 56 bits because 116 bits is the limit to ensure a 128 bit security with $m = 406$. Taking a bigger p_1 leads to a bigger q, that itself leads to $m = 8192$, hence doubling the memory footprint and the computation time.

The performance evaluations of this solution based on arrhythmia classification and image classification case studies is given in Section 2.1.7.

2.1.5.2 Conclusion

Using homomorphic encryption for privacy preserving inference it not always appropriate. As we saw, even for simple network, such as the arrhythmia classification one, it is quite cumbersome to deduce the correct set of parameters so as to obtain decent performance. The naive strategies for the placing the rescaling always leads to the worst performance and memory footprint. For larger networks, the noise growth analysis can be quiet complex to accomplish. As everything is tied together (floating point precision, the noise growth, the size and the number of the primes, the special form of the primes, etc.) is not possible to automatically deduce the best set of parameters; it has to be done by hand. It can be very time consuming. However, on small networks and with good noise growth management it can leads to great performance.

2.1.6 Privacy preserving neural networks using hybrid multi-party computation / homomorphic encryption

In this section we give an overview of the hybrid Multi-Party Computation (MPC) / Homomorphic Encryption (HE) approach for privacy preserving neural network inference. A detailed version is available in the deliverable D3.1.

The classification protocol of GAZELLE [22] combines FHE and MPC (via Garbled Circuits) to compute neural network classifications privately. Fully connected and convolutional layers are computed via FHE. Activation functions and max pooling layers are computed via MPC. Transitions between FHE and MPC are performed by each participant having an additive secret sharing of the intermediate result. This allows to take FHE with very low noise capacity (which results in efficient computation). Transitions between FHE and MPC act effectively as bootstrapping as they reset the noise. Another feature of these transitions is that computational and communication costs grow only linearly with network depth. The transitions between FHE and MPC and the specialized algorithms for linear layers in FHE could be applied regardless of the given cryptographic primitives which realize FHE and MPC. However, another avenue of innovation for GAZELLE lies in the FHE implementation and parameter choice. In GAZELLE, the Brakerski-Fan-Vercauteren (BFV) scheme [16] is used. GAZELLE is secure for semi-honest adversaries, that is, neither the server nor the client recovers any information if they follow the protocol. The protocol does not reveal the weights of each layer or their exact size.

2.1.6.1 Specifications and Implementation

For the Hybrid solution, we have implemented the linear operations such as vector/matrix multiplication using FHE and the non-linear ones such as operations in activation layer by using MPC. On the other hand, we use HElib [2] as the homomorphic encryption library and BGV as the FHE tool. We also employ a truncation method to deal with the real numbers. This method is applied on the plaintext value before and after the classification such that floating point numbers are converted into integers before the classification and the result is converted into floating point number after the classification. See Deliverable 3.1 for more details. The performance evaluations of this solution based on arrhythmia classification and image classification case studies is given in Section 2.1.7.

2.1.7 Performances Evaluation

In this section, we present a performance study that motivates the usage of neural networks in the context of health data and image processing. For this respect, we compare the performance of FHE-based, 2PC-based and Hybrid (GAZELLE-based) solutions for arrhythmia and image classification.

Once these neural networks models are designed, the goal is to execute them over protected inputs. In this section, we present the experimental results to compare the performance of the FHE-based, 2PC-based, and Hybrid solutions on the arrhythmia and image classifications. All the simulations were carried out using a computer which has six 4.0 GHz Intel Core i7-7800X processors, 128 GB RAM and 1TB SSD disk. The experimental results are given in Table 5. We have performed two different tests for classifying of a single heartbeat/image and classifying heartbeats in batches of 2048 heartbeats/images.

For the arrhythmia classification, the 2PC-based solution seems to provide the lowest computational cost when compared with the other solutions for the classification of a single heartbeat. However, the FHE-based solution outperforms when classification is performed in batches. Additionally, the FHE-based solution has better advantage in terms of the communication cost since all the computations in this solution are realized at the server and there is no need for interaction except for the transfer of the input. Moreover, the NN model of arrhythmia classification, which only involves linear operations and one square operation, the FHE-based solution seems to be the most suitable one. Nevertheless, this may not be the case for deeper neural networks such as for the case of image classification. For the image classification, the 2PC-based solution and the hybrid solution outperforms the FHE based solution for the classification of a single image. On the other hand, the FHE-based solution again provides better results for the classification in batches. However, these two NN models are designed to be implemented in all proposed solutions and therefore they do not consist of any non-linear operations such as the max pooling layer and ReLU activation function which are not supported by FHE. The hybrid solution may be the most appropriate one in such a case as it combines the use of 2PC and FHE. Indeed, this particular solution is specifically designed to operate on large NN models and it can be seen that the increase rate on the computational cost is lower than in the case of other solutions. Furthermore, the current version of the hybrid solution does not support packing.

Table 5: Performance evaluation on ECG classification and image classification for FHE-based, 2PC-based and hybrid solutions.

	FHE-based solution (with packing)		**2PC-based solution (with packing)**			**Hybrid solution (without packing)**		
	Comp. Cost (ms)	**Comm. Cost (MB)**	**Online Comp. Cost (ms)**	**Total Comp. Cost (ms)**	**Comm. Cost (MB)**	**Online Comp. Cost (ms)**	**Total Comp. Cost (ms)**	**Comm. Cost (MB)**
1 Heartbeat	1253	3.69	25.7	212.947	1.85	43	5638	15.5
2048 Heartbeats	1253	3.69	3792.7	23092.4	3801	88064	11546624	31744
1 Image	13570	155	205.4	1083.2	3.5	1200	6500	264
2048 Image	13570	155	369878	776778.3	82015	4505600	13312000	540672

2.2 Privacy Preserving Neural Networks Training

In this section, we introduce our privacy preserving neural network (NN) training solution based on Fully Homomorphic Encryption (FHE). The targeted network and accompanying dataset is based on the *CNN-static* network as described in [23] and on the dataset from [27], as published by the Kaggle community [3]

This NN includes an array of connected components, among which are, a Convolution Layer, a Fully Connected Layer, a Dropout Layer, and more. The goal was to determine the number of encrypted training examples that can be used during an 8 hour timeframe to train the encrypted model with encrypted 20-word sentences (each word embedded into a vector of size 100) and 6 output classes, all within a reasonable degradation in classification accuracy.

The training set, that was eventually used, including 4080 samples (extracted from [27]). We were able to use them all during an 8-hour window and reach a small accuracy drop for a two-class classifier and a $\sim$15% accuracy loss for the six-class classifier (both using a 24-CPU machine). Many of the computations involved in training a neural network are highly parallelizable, and so this figure can be further improved as a function of the number of CPUs. In these days, we are working on providing figures based on using more computing power. Thus, we hope to demonstrate that the accuracy-gap can be closed by using an appropriate number of CPUs. Note that since all the network tensor operations involved in the training are done using homomorphic operations on ciphertexts and not on direct matrices, we could not directly use GPUs to speed up this process. However, it may be possible to research into ways of exploiting GPUs for the matrix operations used by the encryption scheme itself, but this does not directly relate to neural network training, and was not done as part of the current research.

The above experiments were performed without the Bootstrapping operation which we are still working on implementing, thus we temporarily rely on the interactive mode, in which every time a ciphertext accumulates too much noise it is sent back to the client, where it is decrypted, encrypted again, and returned. Decrypting cleans the noise and encrypting again creates a fresh ciphertext with minimal noise. Also, certain components that can tremendously improve performance were not yet implemented, among them horizontal packing and other parallelization optimizations.

We believe the results of this 3-month pilot show the potential of this technology and advance it beyond what is currently considered state-of-the-art. We hope to continue this work, improve the performance even further and attempt to productize these capabilities into an asset that would help in solving the customer's analytics challenges.

2.2.1 Neural – Network Simplification

2.2.1.1 Original network

Our goal was to train a given Neural-Network under FHE. The targeted network is the one described by Yoon Kim in [23] and named as *CNN-static*. The network was trained with the accompanying dataset from the CMU Text Learning Group Data Archives [27]. Here, the term *CNN-static* refers to this network as implemented in Kaggle.

[3] https://www.kaggle.com/au1206/text-classification-using-cnn

The **dataset** for the network includes texts originating from 20 different newsgroups (relating to topics such as sports, religion, politics, etc.). Each of these 20 classes is represented by 1000 given texts. The goal of the network is to classify texts of up to 1000 words into these 20 classes. For this purpose, the network is trained with 4/5 of the available texts, and the remaining 1/5 are left for validation.

This **network architecture** is shown in Figure 5. The input texts are first truncated or padded into 1000-word texts. These texts are then **embedded** using a given (previously trained) dictionary that translates every word in the text into a list of 100 (floating-point) numbers.

The *CNN-static* network continues with a **convolutional** layer that includes 3 groups of 512 filters. The first group includes filters that analyze sequences of 3 contiguous words of the text, considering all 100 embedded features of those words. Thus, each such filter is of size 3x100 and analyzes 998 3-word sequences, and results in 998 values. The whole group of 512 such 3-word filters thus results in a matrix of size 998x512.

The 2nd group of 512 filters similarly analyzes 4-word sequences and results in a matrix of size 997x512. The 3rd group of 512 filters similarly analyzes 5-word sequences and results in a matrix of size 996x512. Each value in these 3 matrices is then modified using the non-linear **ReLU activation function**: $ReLU(x) = max(x, 0)$

Now, a **Max-Pooling** layer, “pools” just the maximal value in each column of the 3 matrices, resulting in a vector of 1536 values (512 values from each matrix).

A **dropout** layer randomly drops half of these values (during training only, in order to reduce overfitting). The resulting vector is then passed to a **fully connected layer** that transforms the 1536-value vector into a 20 value vector. The $\mathbf{softmax}$ **activation function** transforms these 20 values into 20 probability values. Finally, the 20 probabilities are compared with the actual labeled data of the input text sample (a “1-hot” vector with all “0”s, except a “1” in the correct location). This comparison results with the final loss value of the model, based on the **cross-entropy loss function**.

Figure 6 gives the accuracy and the loss as they evolve over time. The ‘val_accuracy’ and ‘val_loss’ are the validation accuracy and loss metrics, as measured on the validation set. The ‘accuracy’ and ‘loss’ are the accuracy and loss metrics measured on the training set. The x-axis shows the number of the ‘epoch’ (in each epoch we re-train the model with all 20000 samples, in ‘batches’ of 30 samples).

The accuracy and loss were computed using Keras, a state-of-the-art library for training NNs (in plaintext). The NN was trained using Adaptive Moment Estimation (Adam) optimization algorithm, a common state-of-the-art approach for training NN, usually leading to faster convergence than its simpler variant Stochastic Gradient Descent (SGD).

2.2.1.2 Modifications to original network

The following sections list the modifications that we made to the original *CNN-static* network described above when we designed the network that was actually trained under FHE. The modifications were first tested with plain non-secure training (i.e. not under FHE) in order to make sure that the modifications don’t decrease the accuracy too much. The intermediate accuracy numbers are shown in the figures in the following section. Once all the necessary modifications

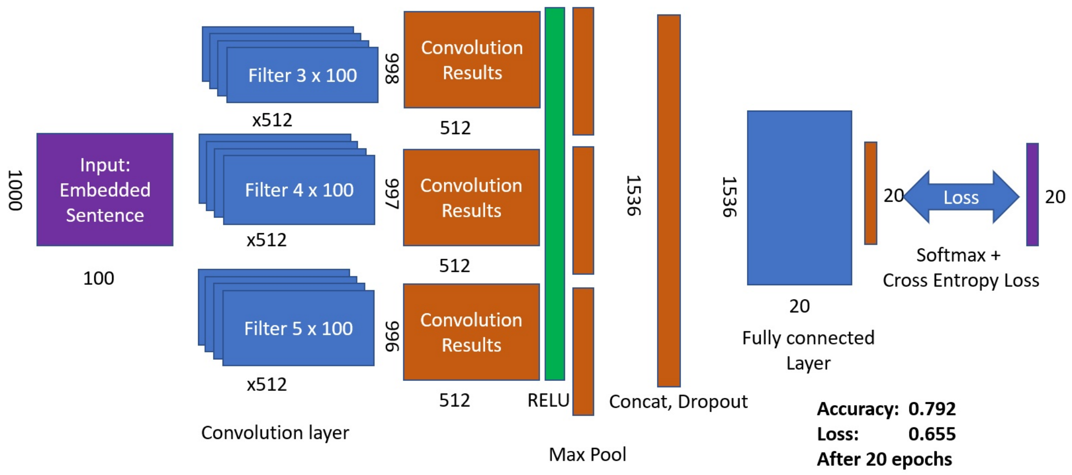

Figure 5: Net-0: the original CNN-static network

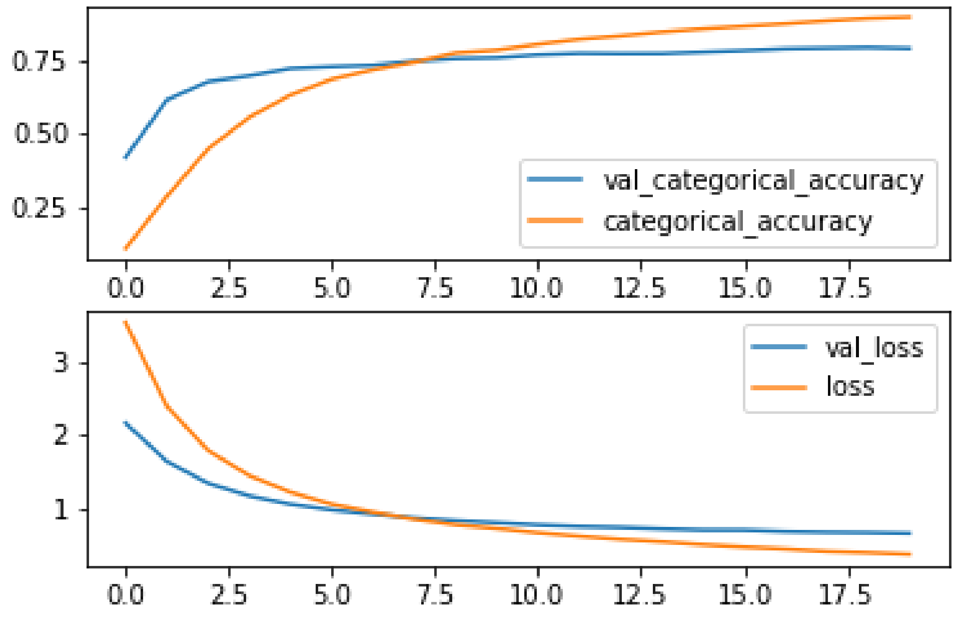

Figure 6: Net-0 accuracy and loss over time

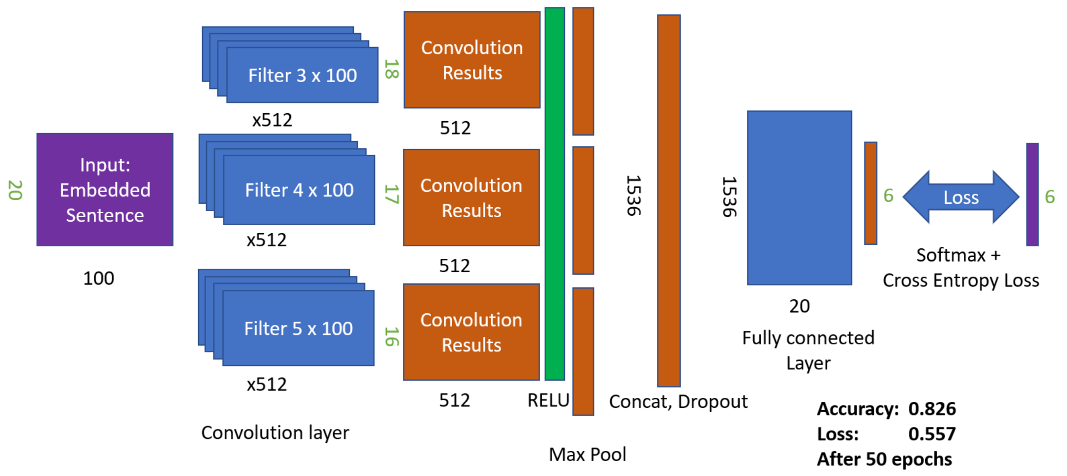

Figure 7: Net-1: 20-word inputs, 6 classes

were made and tested we were finally able to train the modified network homomorphically. The modifications were of the following categories:

- Our goal was to train a network for a somewhat simplified version of the given classification problem, using FHE while optimizing the time necessary for training the dataset from [27]. We modified the classification problem to classify 20-word texts into 6 classes, rather than 1000-word texts into 20 classes as in the original *CNN-static* network. This change also reduced the training set from 20000 samples to 5100 samples, and again we set aside a 1/5 of these for validation (leaving 4080 samples for training and 1020 for validation). Figures 7 and 8 show the resulting network and corresponding accuracy data. This new network now served as the base-line for comparing the plain and the FHE training results.

- The original *CNN-static* network that was described in the previous section includes 625,158 trainable parameters (in the convolution filters and the fully-connected layer). Our target was to train the network in 8 hours. However, the current state of FHE technology is relatively slow and will not be able to train such a large network within the given time constraint. One of our tasks therefore was to **reduce the size** of the network, while approximately maintaining the classification accuracy.

- FHE can theoretically perform any computation on the encrypted data. However, in practice, all available FHE tools support only a very limited set of operations (that can still

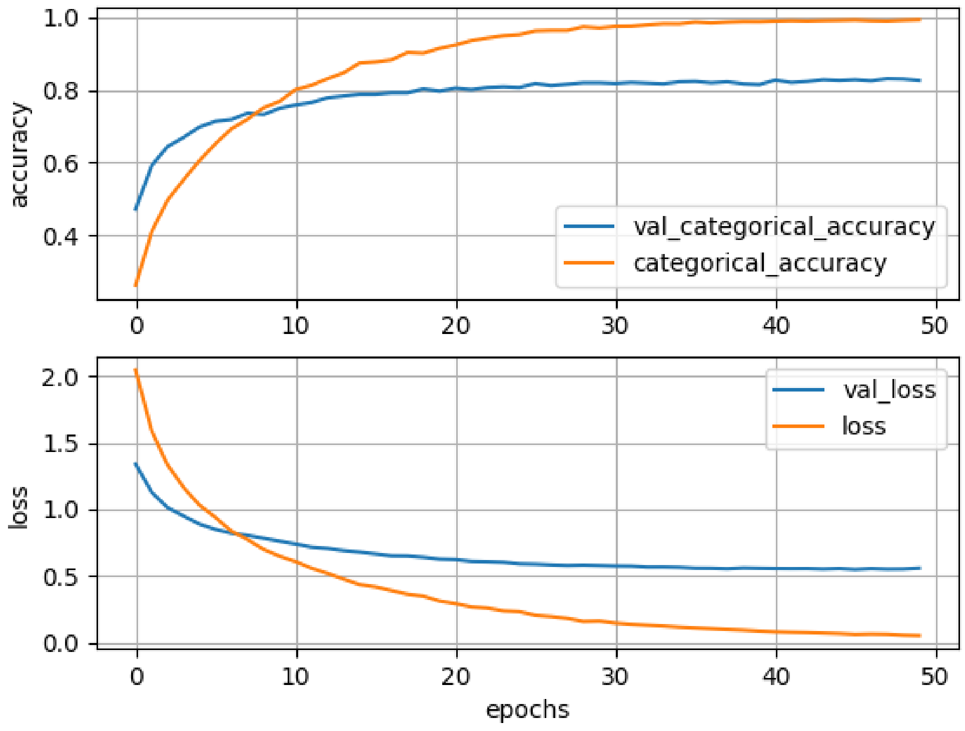

Figure 8: Net-1 accuracy and loss over time

be computed with reasonable performance) generally including member-wise addition, subtraction, and multiplication, and vector rotations.

These limited operations are not enough to perform all the operations needed to perform the training of the *CNN-static* network described above. For example, the ReLU activation function involves a max operation that is not available. The same is true for the Max-Pooling operation, the Softmax activation function (which also involves other unsupported operations), and the categorical cross entropy loss function used by the network (which involves log and conditionals, both unsupported in current FHE platforms).

We should note that all the above "missing" operations can in fact be computed just by using the addition and multiplication operations available in current FHE platforms, for example, by approximating them with high level polynomials. However, high level polynomials involve many dependent multiplications which are costly operations for FHE (because they eventually also require very slow bootstrapping operations to remove the resulting "noise").

Therefore, we replaced the operations that can't be carried out efficiently with current FHE platforms, with **alternative operations** that can be computed efficiently, again - while approximately maintaining the classification accuracy. For example, as will be shown below, we replaced ReLU with a Square activation function, which can be computed since it only involves multiplications, and still provides good accuracy.

2.2.1.3 Embedding

The *CNN-static* network uses an embedding dictionary that translates every word in the text into a list of 100 floating-point numbers. In order to improve the performance of training under FHE, we tried to reduce the network by using an alternative embedding dictionary with only 50 numbers per word. Figures 9 and 10 show the resulting network and corresponding accuracy data. The validation accuracy dropped from 0.826 to 0.773.

2.2.1.4 Activation Functions & Pooling

As mentioned in section 2.2.1.2., current FHE platforms don't support the Max operation. Both the ReLU activation function, and the Max-pooling layer involve a Max operation. We therefore tested the effects on accuracy of replacing the ReLU activation function, with a simple square function (i.e. $activation(x) = x * x$), as this only involves multiplication which is available in current FHE platforms. We also replaced Max-pooling with Mean-Pooling (that only involves additions, and multiplication with a constant – both available in current FHE platforms). Note that after these modifications are made, there is still a non-linear component in the neural network (e.g., the square activation), which is important for learning complex models.

Figures 11 and 12 show the resulting network and corresponding accuracy data. The validation accuracy dropped from 0.773 to 0.761.

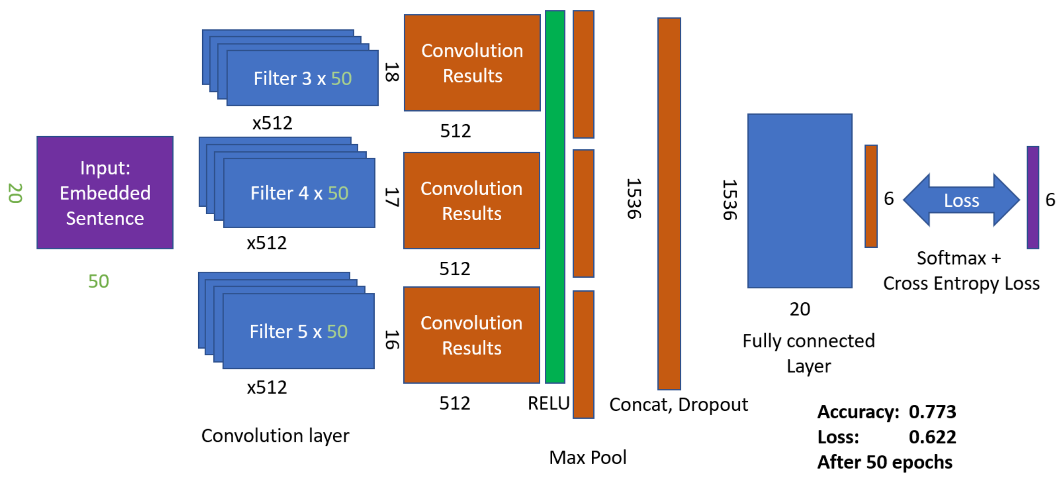

Figure 9: Net-2: 50 embedded features

2.2.1.5 Euclidean Output Function

The *CNN-static* network ends with a softmax activation function followed by a binary-cross-entropy loss function. These operations involve several functions not directly available in current FHE platforms (including max, division, and exponentiation by reals). Following the approach of [33], we replaced these two operations with a Euclidean loss function. This loss function computes the (un-rooted) sum of the squares of the differences of feature values (between the 6 features resulting from the fully connected layer, and the 1-hot labels).

Figures 13 and 14 show the resulting network and its corresponding accuracy. The validation accuracy rose(!) from 0.761 to 0.785. The unexpected rise in accuracy in this run can be explained by the inherent noise observed of about +-0.02 in the accuracy, due to random factors, like initializations, dropout, etc.

2.2.1.6 SGD & Filter reduction

The *CNN-static* network uses the Adam optimization algorithm, which involves performing division by the variance of the gradients during backpropagation. Since current FHE platforms don't support division, we replaced Adam with SGD (Stochastic-Gradient-Descent) with momentum (a simpler known extension to plain SGD). Our tests demonstrated that the momentum component does not improve the accuracy of the network, so we ended up using simple SGD, with a learning rate of 0.01.

Also, we further reduced the dimensions of the network (to improve run-time performance) by

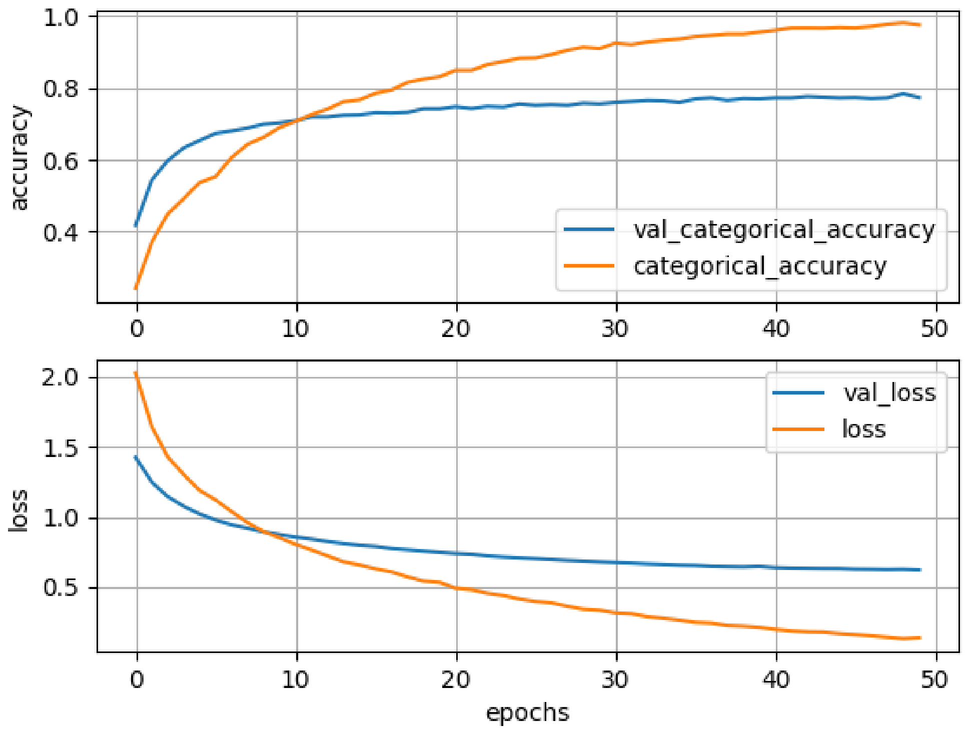

Figure 10: Net-2 accuracy and loss over time

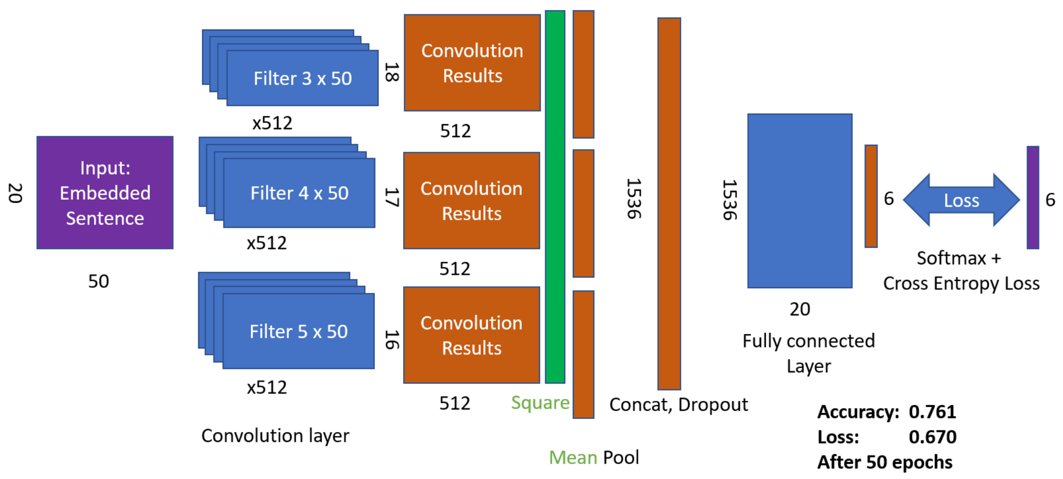

Figure 11: Net-3: square activation, mean pooling

dropping many of the convolution filters: we dropped the entire group of 4-word filters and reduced the other two groups to just 100 filters (instead of 512). Finally, we found that decreasing the drop rate of the dropout layer to 0.1 (instead of 0.5) improves the validation accuracy.

Figures 15 and 16 show the resulting network and corresponding accuracy. The validation accuracy again rose from 0.785 to 0.790. Again, the unexpected rise in the final accuracy in this run can be explained by the inherent noise observed of about +-0.02 in the accuracy. Also, note that this time it took the accuracy more epochs to stabilize (after ∼75 epochs, rather than only after ∼50 epochs).

2.2.1.7 Batch size & final parameter adjustments

As can be seen by the above graphs, the accuracy reaches good results only after about 75 epochs. Recall that in each epoch we re-trained the model with all (4080) samples, in 136 “batches” of 30 samples. 75 such epochs would take too long with our FHE based system, which (at the time of testing) took ∼2 minutes to train each batch. Each epoch included 136 batches of 30 samples, thus the training would complete less than 2 epochs in the targeted 8-hour timeframe.

A feature of the current FHE based training architecture (see next chapter) is that the size of the batch does not affect the time that it takes to train it, as long as it’s below 4096 samples (the size of the cipher vector). It thus makes sense to train more than 30 samples in each batch. There is however an upper limit to how large a batch can be, since the training set includes

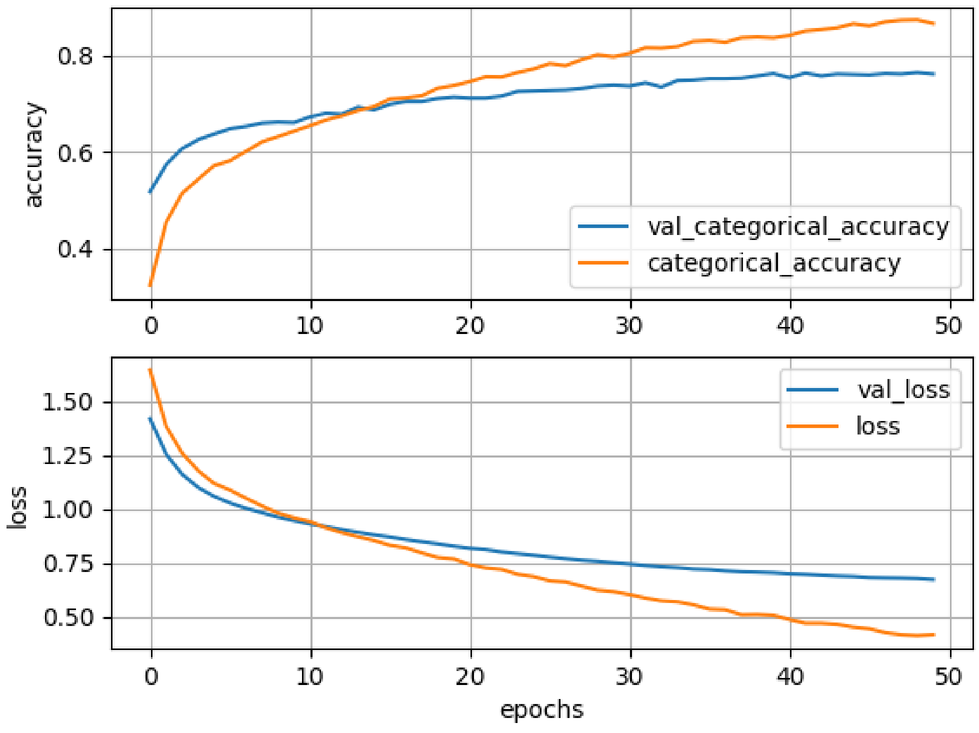

Figure 12: Net-3 accuracy and loss over time

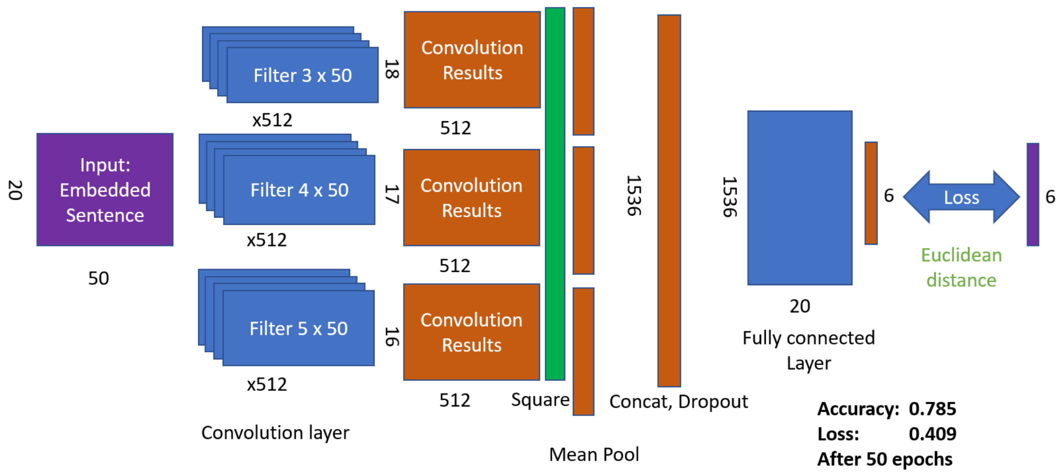

Figure 13: Net-4: Euclidean based loss

only 4080 samples. We tried several batch sizes and reached good results with 16 batches of 255 samples per epoch. A further decrease in the number of filters (from 100 to 25 per filter type) and an increase in the learning rate (from 0.01 to 0.05) lowered the final accuracy from 0.790 to 0.752 but enabled the FHE system to train for less than 8 hours on a 64-CPU machine and reach the 37'th epoch with 0.66 validation accuracy. Within $\sim$13 hours it reached the 64'th epoch with 0.72 validation accuracy. Recall that the original network (simplified to classify 20 word sentences into 6 classes - see section 2.2.1.2) reached a final accuracy of 0.826. Figures 17 and 18 show the resulting network and its corresponding accuracy data.

2.2.1.8 Future research

Several points were raised during our work that require further research. One issue is related to the usage of dropout. We implemented two variants of dropout. One is per-batch, which means that a neuron randomly selected for dropout is consistently dropped out for the entire batch. The second is per-sample, which means that for each sample in the batch we independently and randomly select which neurons are dropped out, so a neuron dropped out for one sample in the batch may not be dropped out for another sample in the same batch.

In all our experiments we used the per-sample variant which is more common. The per-batch variant was preferred because it has a time performance advantage, even though some initial experiments showed that it results in somewhat inferior accuracy. Dropping out a neuron for the entire batch allows shutting down whole sections of the network during batch processing and

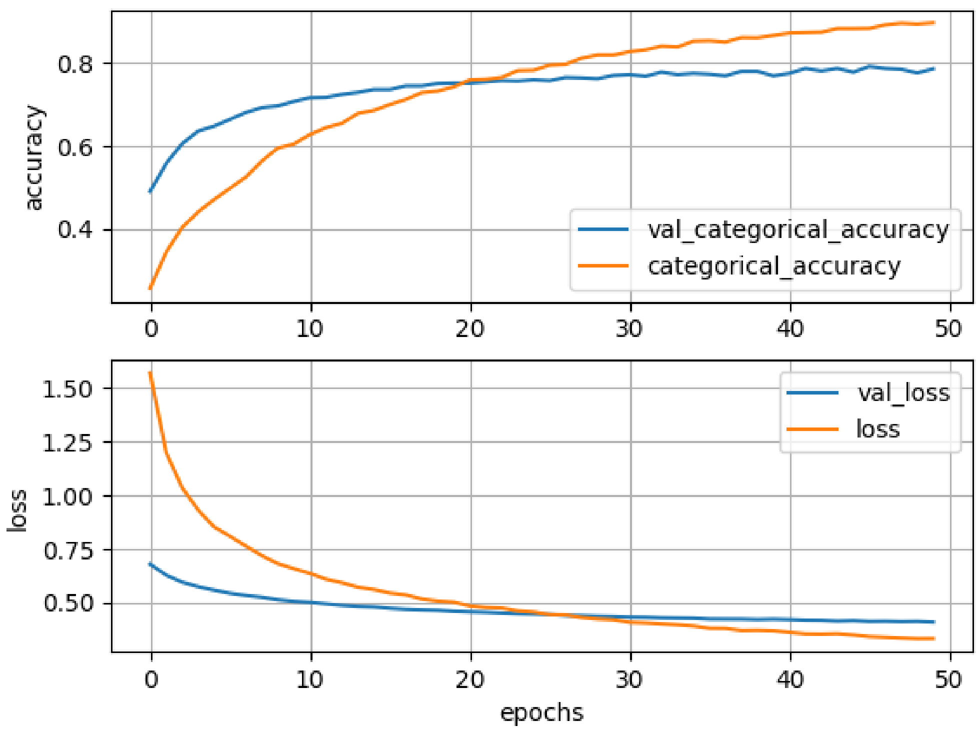

Figure 14: Net-4 accuracy and loss over time

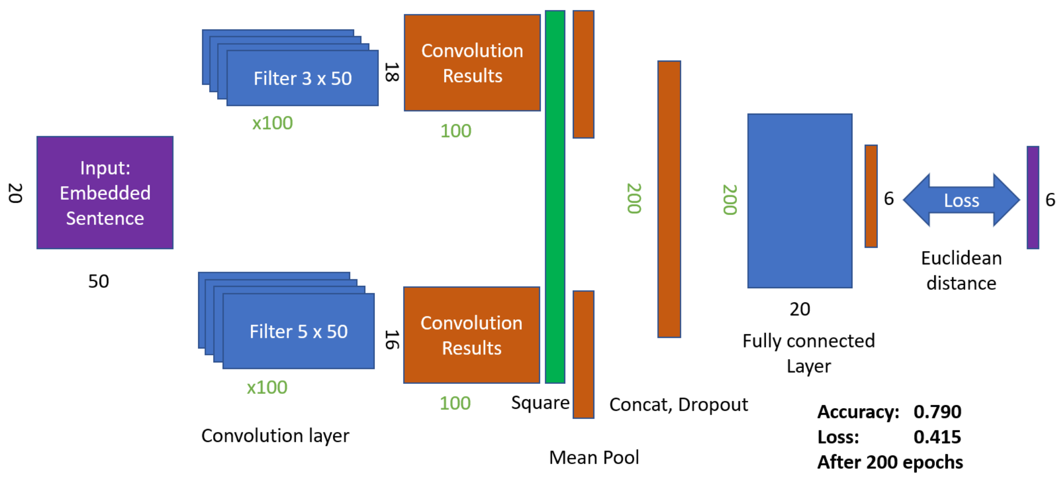

Figure 15: Net-5: SGD, reduce filters, parameter adjustments

achieve a non-trivial boost in performance. In the future we plan to check whether per-batch dropout can be tuned to provide similar results as in the per-sample variant, so as to enjoy its better efficiency.

Another issue is related to batch size. Working with large batch sizes has an advantage when working under homomorphic encryption, since usually ciphertexts carry a large number of 'slots' that are not always easy to utilize efficiently with small batch sizes. On the other hand, large batches may make the training process inefficient, and require additional epochs. Finding the optimal balance is another task for future research.

A third issue is regarding the order between the mean pooling and activation layers that come after the convolutional layer. Currently the activation layer precedes the mean pooling layer, but a different order may yet again provide a time performance boost, since applying mean pooling directly on the convolution results may allow computing both these layers together in a more efficient manner.

Finally, we would like to explore using batch normalization in order to improve numerical stability. Reducing network size, increasing batch size, and increasing learning rate, all have positive effects on time efficiency, but negative effects on numerical stability. Batch normalization preprocesses the data prior to feeding it to a network layer, and may improve numerical stability, thus allowing us to further optimize the training process.

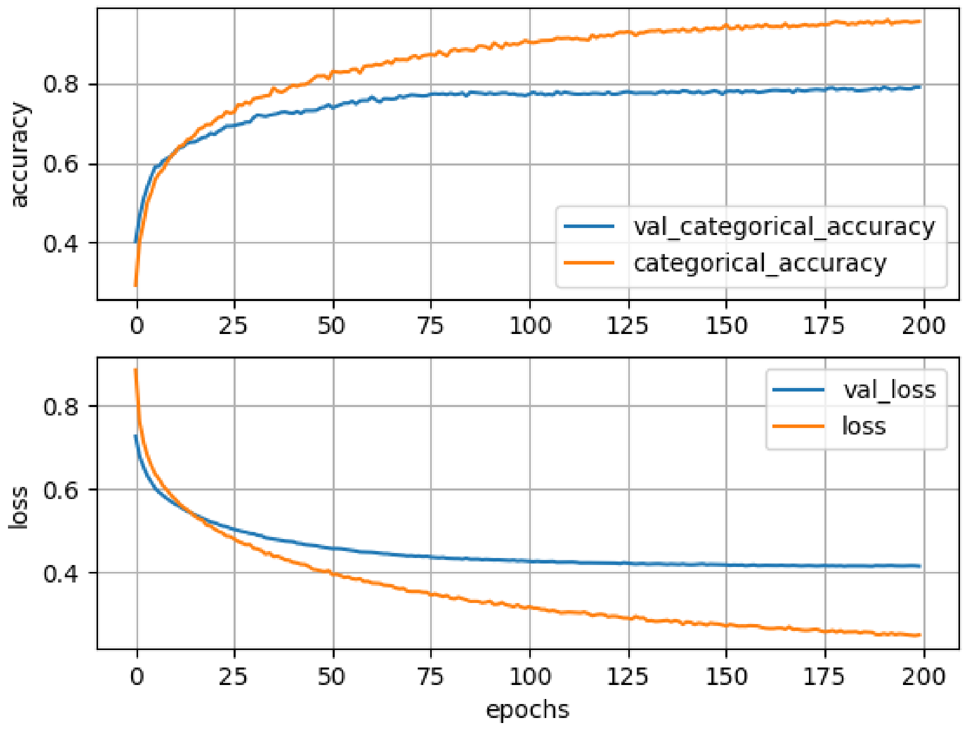

Figure 16: Net-5 accuracy and loss over time

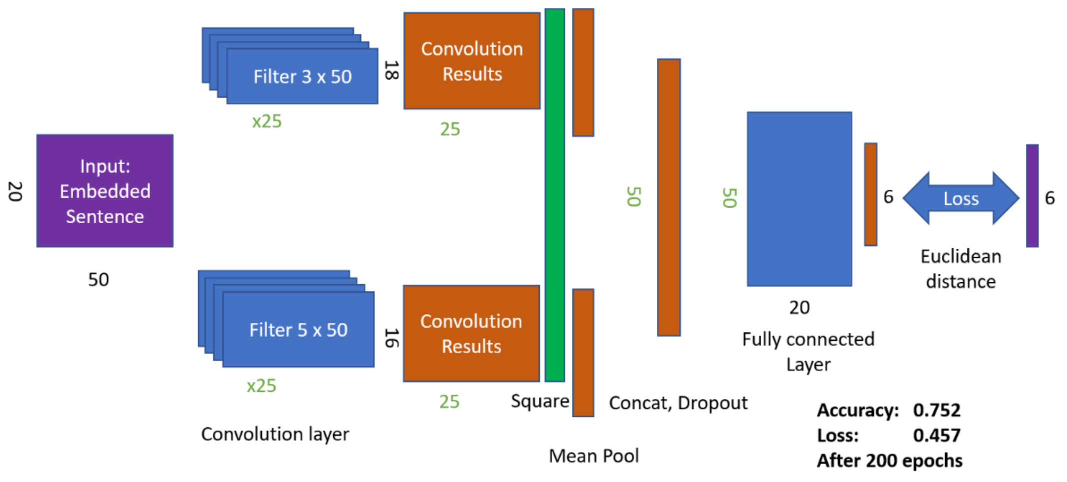

Figure 17: Net-6: final network, further parameter adjustments

2.2.2 Implementation and Performance

2.2.2.1 Architecture overview

We implemented our code in C++, providing an executable utility that reads input data encrypted under FHE, and outputs a trained NN model encrypted under FHE. The encryption's strength is equivalent to a 128-bit secret key.

Working with HE required from us to write standard algorithms for Neural Network inference and training from scratch, since the requirements of working efficiently with HE requires the complete redesign of the code, and standard libraries such as Keras cannot be adapted to such changes. However, we took special care in making sure that the algorithms we write are equivalent to standard NN libraries using automated tests.

We implemented the following standard components of a Neural Network: a convolutional layer, a fully connected layer, square activation, mean pooling, dropout, and a Euclidean loss function. Our framework allows to combine these components almost freely in any way, as in standard NN libraries. For training, we further support working with a customized batch size of up to 4096 samples per batch, and an optimization algorithm of either simple SGD (Stochastic Gradient Descent) or a version of SGD with momentum that is adapted for efficiency under HE.

Working with HE gradually adds noise to the encrypted data, and this noise should be cleaned every once in a while. This is the purpose of the bootstrapping operation, which is currently not yet implemented for the CKKS scheme used by our platform. In the meantime, we use a placeholder operation of decrypting and re-encrypting using the known secret key, which

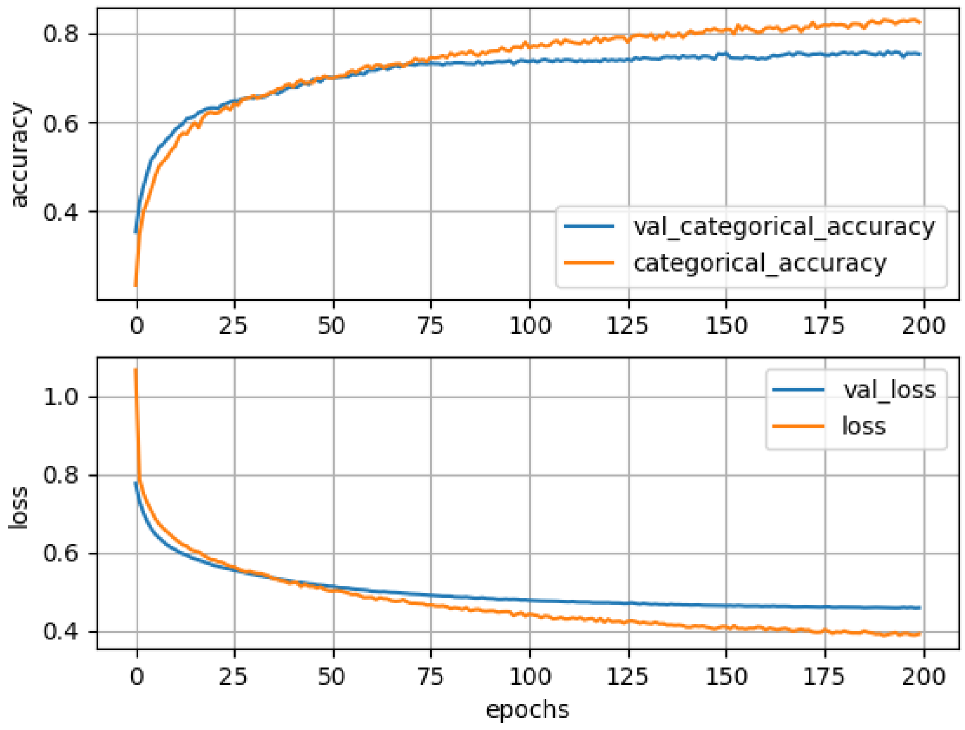

Figure 18: Net-6 accuracy and loss over time

has the same noise cleaning effect. Our placeholder operation is much cheaper and simpler to use than a real bootstrapping operation, but we count the number of times that it is required in order to be able to estimate the performance after its expected future replacement with the real operation.

2.2.2.2 Packing and optimization

As usual in FHE, each ciphertext contains a preconfigured number of slots, and each FHE operation acts on all slots in parallel. We configured our platform to use 4096 slots, as this was the minimal value that gives both 128-bit security and the ability to perform the required depth of operations. Thus, each homomorphic operation we use acts in parallel on all 4096 of the ciphertext's slots.

Efficiently packing the input data, the network's parameters, and the vectors flowing forward and backward during training is challenging, as operations such as matrix multiplication requires carefully arranging and rearranging the data. A homomorphic operation in which not all the slots are used wastes computational power.

We chose a packing method which we called *vertical packing*, in which each slot contains a different input sample, and the network parameters are effectively duplicated 4096 times. This allows us to work in two alternative modes which we call *stacking* and *batching*. In the stacking mode we divide the training set into 4096 distinct groups, and effectively train 4096 separate models that can later be combined using voting. This method is only effective for a large training set (in the order of millions). Our experiments were based on a much smaller training set, and so we used the batching mode, which is somewhat less efficient, but is equivalent to the standard way of training a single model.

A drawback of this approach is that it reaches peak efficiency for a batch size of 4096 samples, whereas typical batch sizes, as is also true in our case, are in the order of 30 samples. We are currently implementing a different packing method with is expected to have peak efficiency at 256 samples.

Many of the computations involved in training a neural network are highly parallelizable, allowing us to parallelize the computation using multithreading.

2.2.3 Results

In section 2.2.1 we described a sequence of modifications and simplifications that we made to the original *CNN-static* model, while approximately maintaining classification accuracy. In parallel with this process we also tested a series of networks and learning configurations to optimize the overall performance that results when training with FHE using the architecture described in the subsections above. We were thus able to identify and resolve some performance bottlenecks (like reduction in the number of CKKS relinearizations, optimal use of threads, etc.).

The original target of our research was to optimize the training time per sample (as amortized from the total training time for all the epochs). We later made the reasonable modification of this goal and targeted the achievement of the best possible accuracy in the allowed 8 hours of training under FHE. As seen below, the final result was 0.66 validation accuracy, with 47 sec per batch (on a 64-CPU machine).

Table 6 shows the accuracy and performance results reached for 4 different system setups using two machines - (1) an Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz machine with 24 CPUs and 64 GB memory, and (2) an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz machine with 64 CPUs and 216 GB memory. Machine (1) was used in all the setups except in setup-3 which used machine (2). In all cases, the training set (from [27]) included 4080 samples, plus 1020 samples kept separately for validation.

In **setup-1** we split the entire training data into just two batches (of 2040 samples each). We are using the batching mode of the vertical packing scheme (see section 2.2.2.2), where the computations takes the same time for small and large batches. Thus it may make sense to use as large a batch as possible. However, in this setup the batch includes half of the available samples, which is not an ideal setup for training. Using such a large batch may converge better when much more data is available in the training set. Also, this setup includes 200 convolution filters and 41,406 trainable parameters which slows down the training of each batch. This results in less batches trained within the allowed 8-hour timeframe, and thus lower final accuracy.

In **setup-2** we reduced the batch size in order to increase the number of batches in each epoch, and also reduced the number of filters and trainable parameters. This resulted in less progress made after each batch, but the deficiency was compensated by the much larger number of faster batches. Thus, this setup gave better accuracy after 8 hours.

In **setup-3** we ran the same configuration as in setup-2 but on hardware with twice as many CPUs (64 vs. 32) and with more memory (216 GB vs. 64 GB). This enabled us to reach twice as many epochs during the allowed 8 hours and somewhat better accuracy.

In **setup-4** we ran again on the smaller 32 CPU machine and with the original high number of trainable parameters, but we reduced the problem size by classifying samples from only 2 (rather than 6) newsgroups. This is an easier problem with a smaller training set, allowing us to reach an accuracy of 0.876 in a little over four hours

2.2.4 Extrapolating to larger networks

The performance of training a NN of course heavily depends on its size, depth, and structure. The vertical packing method we used for all results included in this report easily and linearly scales when increasing the network. So, for example, if a single fully connected layer of size NxM costs 1 second in computation time for each iteration, and 10 bootstrap operations, then D such layers would roughly cost D seconds and 10D bootstrap operations. Similarly, increasing a single such layer to size $2N$xM or Nx$2M$ would roughly double its cost ($2N$xM will also double the number of bootstrapping operations, but Nx$2M$ will not, for various technical reasons).

As mentioned above, however, vertical packing is generally not the most efficient. We are now working on a new kind of packing cleverly optimized for the given NN architecture as described in this report. This new kind of packing currently requires specific optimizations and tuning for a given architecture, so adding more layers may require re-optimizing it. We believe it would still exhibit roughly linear scaling properties.

Table 6: Accuracy and performance results for 4 different setups

	setup-1	**setup-2**	**setup-3**	**setup-4**
CPUs	24	24	64	24
Total training time	7hr 14min	8hr 8min	7hr 43min	4hr 20min
Final loss	0.826	0.587	0.53	0.284
Final accuracy	0.285	0.609	0.66	0.876
epochs	16	15	37	32
batch size	2040	255	255	1360
batches/epoch	2	16	16	1
batches	32	240	240	32
learning rate	0.01	0.05	0.05	0.01
convolution filters	200	25	25	100
trainable parameters	41406	10356	10356	41406
bootstrapping operations	230784	241440	547264	128192
bootstrapping/batch	7212	1006	1006	4006
Time to train a batch	13.5min	2min	47sec	8.1 min
Time to train an epoch	27min	32min	12.5min	8.1 min

2.2.5 Conclusions

FHE holds great potential in facilitating privacy preserving analytics in complex use cases. During this PoC we were able to train a complex NN within an 8-hour timeframe with an accuracy degradation which is linearly dependent on the number of CPUs used. Currently, for 6 output classes and 24 CPUs the accuracy drop is about 20%, and for 2 output classes there is hardly even an accuracy drop. Due to the parallelization work that was done, this figure can be improved by increasing the number of CPUs. In these days we are working on providing figures that are based on using more computing power. Thus, we hope to demonstrate that the accuracy-gap can be closed by using an appropriate number of CPUs.

Certain components that can tremendously effect performance were not implemented. Among them are, Bootstrapping, Horizontal packing and other parallelization optimizations. We aim to improve the performance of the proposed solution even further and attempt to productize these capabilities into an asset that would help solve the potential customers' analytics challenges.

2.3 Privacy Preserving Collaborative Training

In this section we describe the implemented approaches of privacy preserving collaborative training, which is useful for use case 2 (see deliverable D2.1). We review the achieved results of each approach and describe pros, cons, the discovered gaps and future work. In addition we describe the work that has been done in the aspect of defense against adversary participant. We describe the attack that we chose based on our use case setting relevance, attack's evaluation, proposed defense and its evaluation.

2.3.1 Specifications and Implementation

As we mentioned in the deliverable D3.1 we implemented an approach presented in [37], we evaluated on the Swell dataset [25]. The main objective of the collaborative training is a stress detection. The evaluation performed in two settings with Differential Privacy noise and without. We performed this training task with the following settings:

2.3.1.1 Collaborative training without Differential Private noise

- We used only 10% of the data, 36K samples
- 3 participants
- each of the participants received $\sim 3\%$ of the data, 12K samples
- Upload fraction 0.5 (each of the participants uploaded 50% of the gradients)
- Download fraction 0.4 (each of the participants downloaded 40% of the model's weights)
- All participants upload the gradients and download the model after each epoch

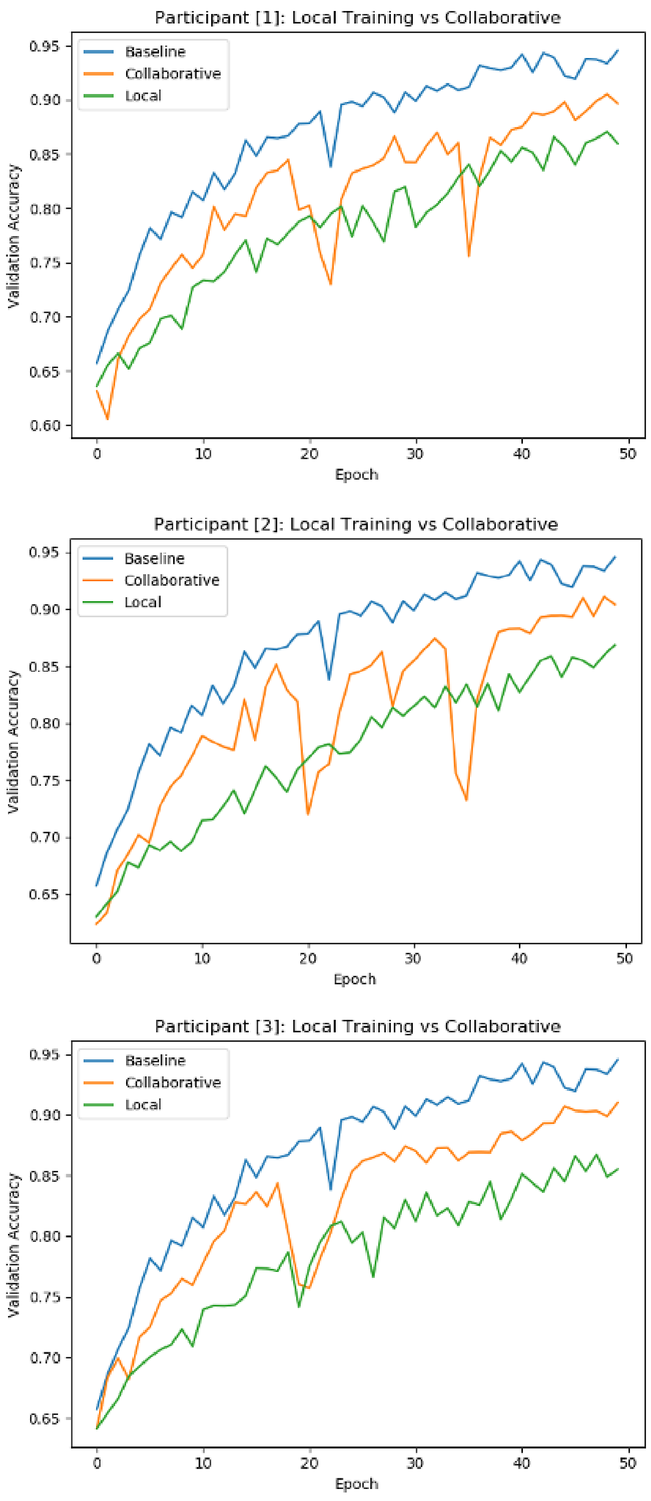

Figure 19: Validation Accuracy, three participants

The achieved results:

The Figure 19 presents a comparison of validation accuracy for each participant between three setups: baseline, local and collaborative. Baseline result is defined when the NN trained on the centralized dataset (10%). Local result presents a setup when each of the participants performs training only on a local data and the Collaborative result when all three participants perform collaborative training.

As expected, the collaborative approach exceeds the results of training on the local data only, and less accurate than the baseline. The advantages of the collaborative approach can be seen starting very first epochs. By sharing only part of the gradients and leaving the participants' data on their premises, some privacy has been achieved. However, there is no way to quantify the privacy guarantee.

2.3.1.2 Collaborative training with Differential Private noise

As the second setup, we evaluated the method with differential private noise based on approach in [37]. In addition we added averaging functionality to the server. The server aggregates updates from each participant and averages the value of the updates based on the number of participants. We performed this training task with the following settings:

- We used only 10% of the data, 36K samples
- 3 participants
- each of the participants received $\sim 3\%$ of the data, 12K samples
- Upload fraction 0.1 (each of the participants uploaded 10% of the gradients)
- Download fraction 0.1 (each of the participants downloaded 10% of the model's weights)
- All participants upload the gradients and download the model after each epoch

The achieved results:

The Figure 20 presents a comparison of validation accuracy for each participant between three setups: baseline, local and collaborative. Baseline and local settings are similar to the defined above, when in the collaborative training each of the participants added an differential private (DP) noise to the uploaded gradients.

As expected, the collaborative approach exceeds the results of training on the local data only, and is almost as accurate as the baseline. However, the convergence time is increased due to added noise and updates averaging performed by the server. We choose to share and download only a small part of the gradients/weights in order to reduce the global DP ε. In this approach ε is a function of:

$$\varepsilon \simeq \theta_u * M_w * p$$

While:
θ_u - gradients upload fraction

M_w - number of model parameters
p - per gradient privacy budget
This approach provides a good privacy guarantee per models parameter (per gradient) and prevents information exposure regarding a specific gradient. However, the total privacy guarantee is $\varepsilon = 7 * 10^5$ which by definition has no privacy guarantee with this value.

2.3.1.3 Training local differential private model and weights sharing

Since in the first approach the privacy guarantee value is meaningless in the aspect of DP definition, we decided to check an additional approach presented in [4]. This approach presents a way to train a NN model, which is supposed to be released, without exposing anything regarding the training data of this model. In addition, [4] presents a new privacy guarantee accountant function, which provides a tighter epsilon calculation. This approach is implemented and published as a library, TensorFlow Privacy[4]. There are some Federated learning approaches that employs this approach. However they are not aligned with our use case. Mainly, they require a large number of participants e.g. 10^5 and trusted server. Thus, we had to try to adopt this approach in the different way. We adopt this approach in the following way:

Algorithm 2: Local DP model and weights sharing

```
Client Side: Each participant performs the following steps
repeat
for each training epoch $t$ do
    Download the centralized model from the central server and replace the local model
    Train a local differential private model $M_{t+1}$ on a private training data
    Compute local model changes (gradients)
        $\theta_{t+1} = M_{t+1} - M_t$
    Upload the gradients $\theta_{t+1}$ to the central server
end
until the desired accuracy is achieved, or when the accuracy is not improved anymore
The server performs the following steps:
On upload:
    • Given a gradient vector $\theta$, add the gradients to the global model
    • $G = G + \frac{\theta}{\#participants}$
On download
    • Return the global model's parameters $G$.
```

[4]https://github.com/tensorflow/privacy

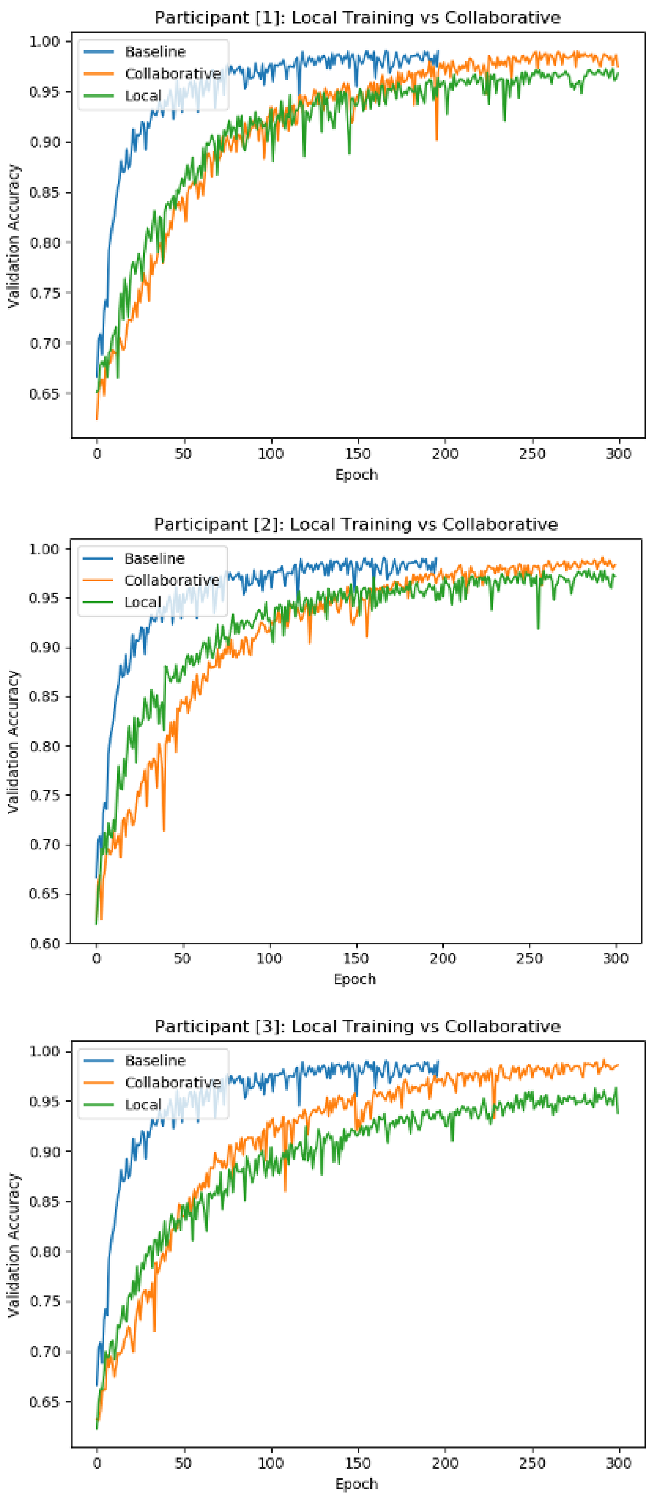

Figure 20: Validation Accuracy, three participants

The achieved results:

As an initial step, we evaluated this approach on the MNIST dataset. After trying different DP parameters, our decision was, that on a not trivial NN topology this approach has a major degradation on the network's utility. Even with an epsilon $\varepsilon = 3800$ the training on the local data only achieved higher validation accuracy than the collaborative training.

2.3.1.4 Conclusion

In this section we describe our conclusion based on two implemented approaches.
Privacy-Preserving Deep Learning based on [37].
Cons:

- Privacy per gradient approach does not provide theoretical proof for strong privacy guarantee for the entire model
- For non trivial NN models achieved a large ε. However, we are not aware of any membership inference attack on this specific approach. Thus, the efficiency of the attack needs to be checked empirically.

Pros:

- There is a significant accuracy improvement when using a collaborative training
- Can be applied on complex NN networks

Collaborative Training using local DP based on [4].
Cons:

- Applying local DP on the non trivial model has a significant impact on the model utility. Sometimes a model does not converge for large NN, even without applying a collaborative aspect of the approach.
- For a model with non trivial topology, model simplification should be performed in order to apply this method.

Pros:

- The approach provides tight privacy guarantee

2.3.2 Defenses against adversary participant

As mentioned in the previous deliverable D3.1, there are different attacks on collaborative training settings. Assuming semi-honest participants, some of the attacks might not be relevant. We have investigated and applied a defense against property inference attack presented in [32], which can be most relevant for our use case. The [32] presents variety type of attacks. In some of them the attacker observes the embedding layer in order to infer existence of some specific word or sentence in the victim's dataset. Since the NN topology in our use case does not have an embedding layer, we address a more general approach presented in the paper [32], which allows adversary participant to infer uncorrelated, to the main task, features.

2.3.2.1 Property inference attack

The attack is based on the observation that the NN model learns features which are not necessarily correlated with the main task objective. The attacker tries to infer whether the victim's dataset contains data samples with the uncorrelated property or not. For example, when the main task objective is gender classification, the attacker can infer whether the victim's dataset contains data samples of peoples wearing eyeglasses or not. We assume that such property can be sensitive. We assume that the attacker has an auxiliary data with the property he wants to infer. In addition, we assume that the attacker can isolate victims' gradients upload. The attacker meets the criteria of semi honest but curious participant.

The attack flow:

Algorithm 3: Honest participants' and victim's actions

```
repeat
for each training iteration $t$ do
    Download the centralized model from the central server and replace the local model
    Train a local NN model on a private training data
    Compute local model changes $\theta_t$ (gradients)
    Upload the gradients $\theta_t$ to the central service
end
until the desired accuracy is achieved, or when the accuracy is not improved anymore
```

Algorithm 4: Attacker's actions

```
During the collaborative training phase
Train a local NN model
if $attacker\_mode = ACTIVE$
then:
$Loss = \alpha * Loss(main\_task) + (1 - \alpha)Loss(property)$
else:
$Loss = Loss(main\_task)$
Upload gradients to the server
Download the updated model $M_{latest}$ and distill victim's gradients $\theta_{vic}$
$D_{vic} \leftarrow D_{vic} + \theta_{vic}$
for $b \in D_{aux\_prop}$ do
    $D_{train\_prop} \leftarrow D_{train\_prop} + gradients(M_{latest}(b))$
end
for $b \in D_{aux\_non_prop}$ do
    $D_{train\_non\_prop} \leftarrow D_{train\_non\_prop} + gradients(M_{latest}(b))$
end
Attack phase
Train $M_{attack}$ on $D_{train\_non\_prop} \cup D_{train\_prop}$
Classify $M_{attack}(D_{vic})$
```

During the training phase the attacker collects victim's gradients data D_{vic} and creates training dataset for the attack model, using auxiliary data labeled with relevant property. Data with property defined as batch gradients when at least one data sample in the batch has a property. We evaluated this attack using LFW [21] dataset, pictures with people faces. The main task was gender classification and the sensitive property we defined as people who are wearing eyeglasses. As the attack model M_{attack} we chose a Random Forest, which is presented in the paper as a model with the best performance.

2.3.2.2 Defense

The main idea of our defense is to cause the maximal error on the property classification. In order to do so, we defined that a victim trains a local model on multi tasks. The first task is the main task and the second task is to maximize the error of the property classification. We implemented this using adversarial loss. Victim trains his local model using:

$$Loss = (1 - \alpha) * Loss_{main} - \alpha * Loss_{property}$$

By minimizing this Loss function, we minimize the error on the main task and maximize the error on the property task. The main and single assumption that we assume is that the victim has a correct property labeled data.

2.3.2.3 Results and Conclusion

Table 7: Adversary loss VS attack accuracy (2 participants, 20% data with property)

α	**attack accuracy**	**main task accuracy**
0	81%	90.74%
0.05	78%	82.60%
0.1	72%	80.3%

Table 8: Number of participants VS attack accuracy ($\alpha = 0$, 20% data with property)

participants	**attack accuracy**	**main task accuracy**
2	81%	90.74%
3	41%	91.26%
4	37%	91.26%

In the tables 7, 8 and 9 we see that indeed usage of adversarial loss decreases the accuracy of the attack. However, there is a tradeoff between the attack prevention and main task accuracy. In addition, we observed the following behaviors:

- As the number of participants increases, the success of attack decreases

Table 9: Percentage of victim's data with property VS attack accuracy ($\alpha = 0$, 2 participants)

$\%$ **of data with property**	**attack accuracy**	**main task accuracy**
50%	85%	90.74%
20%	81%	90.74%
10%	80%	90.66%

- As the percent of the sensitive data in victims' dataset decreases, the attack accuracy decreases as well

The combination of our defense and alignment in collaborative training setup based on our observation, can mitigate the success of the attack.

3 Privacy Preserving Trajectory Clustering

In this section, we describe two solutions for privacy preserving trajectory clustering. The first one is based on a modification of TRACLUS algorithm [28] in Subsection 3.1. The second one is based on a variant of the MinHash algorithm [7] in Subsection 3.2.

3.1 Privacy Preserving Trajectory Clustering Based on TRACLUS

3.1.1 Goals

In this section we investigate the problem of privacy preserving trajectory clustering that may be useful for use case 3 (UC3) (see deliverable D2.1). We consider a scenario whereby a data owner DO holds a dataset D of trajectories and would like to delegate the trajectory clustering operations to a third-party, untrusted cloud server S (such as the PAPAYA platform). S performs the clustering algorithm over the received dataset and sends the resulting number of clusters with their respective sizes accordingly to DO. In our setting, S is also considered as potentially malicious. Therefore, while delegating the clustering operations to the cloud server, the newly developed privacy preserving trajectory clustering should not leak any information about the actual dataset to any external party and the cloud server itself.

As described in deliverable D3.1, trajectory data can be analysed using a dedicated clustering algorithm named TRACLUS [28]. This algorithm consists of identifying clusters of line segments (segments belonging to trajectories) using an appropriate distance metric and two threshold parameters that are used, on the one hand, to compare distances to determine line segments' neighborhoods and, on the other hand, verify if the number of neighbors is sufficient to create cluster. Similarly to the case of neural networks, in TRACLUS, some operations are too complex in order to be compatible with advanced cryptographic tools such as homomorphic encryption or secure multi-party computation. In order to perform these computations with cryptographic tools and therefore ensure data privacy, some operations need to be either transformed or approximated.

Therefore, the goal of a privacy preserving variant of trajectory clustering is to develop an approximated version of the original clustering algorithm and combine it with cryptographic tools in order to identify these clusters without the server discovering any information on the dataset and while resulting clusters remain consistent. In the context of trajectory clustering, the silhouette [35] analysis seems to be an acceptable method to evaluate/validate the consistency of the actual clustering algorithm.

In the next section, we describe the newly developed privacy preserving variant of TRACLUS (named as ppTRACLUS) that is based on the use of two-party computation.

3.1.2 Specifications and implementation

TRACLUS Overview. TRACLUS consists of first segmenting trajectories into smaller line segments and identifying clusters based on these line segments using an appropriate distance metric. In TRACLUS, the distance between two segments $dist(L_i, L_j)$ is defined as a weighted sum of three components $dist(L_i, L_j) = w_\perp.d_\perp + w_\parallel.d_\parallel + w_\theta.d_\theta$ where $d_\perp$, $d_\parallel$ and d_θ represent the perpendicular, parallel and angular distances, respectively, and $w_\perp$, $w_\parallel$ and w_θ their corresponding weights. The setting of the weights depend on the actual case study. Figure 21 illustrates the computation of the distance metric for the original TRACLUS algorithm.

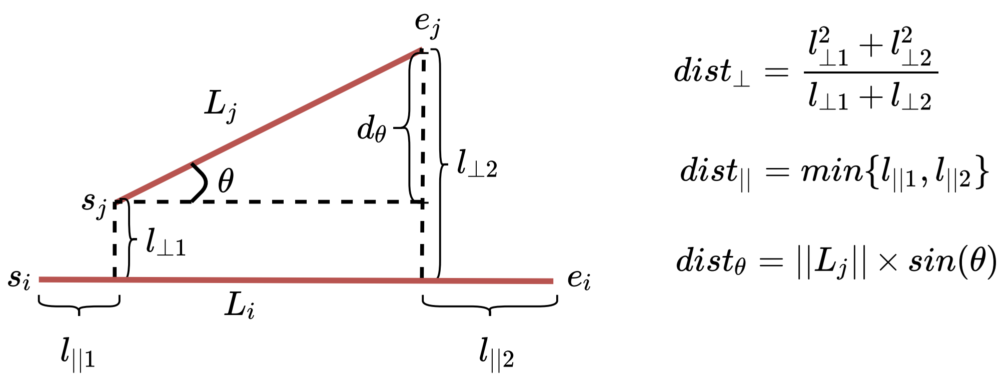
e_j
L_j
d_θ
$l_{\perp 2}$
θ
s_j
$l_{\perp 1}$
s_i
e_i
L_i
$l_{||1}$
$l_{||2}$
$dist_\perp = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}$
$dist_{||} = min\{l_{||1}, l_{||2}\}$
$dist_\theta = ||L_j|| \times sin(\theta)$

Figure 21: Line segment distance function in TRACLUS [28]

The distance between each pair of line segments is first computed and then compared with a given threshold ϵ in order to regroup them in a preliminary cluster. Additional comparisons are performed to check if, this line segment has sufficient neighbors within the actual preliminary cluster. The threshold for the number of neighbors is as $MinLns$. In this case, the cluster is validated and further comparisons are similarly performed with bordering line segments for cluster expansion purposes.

As previously mentioned, the original TRACLUS algorithm needs to be modified in order to be able to support cryptographic tools. Indeed, the computation of the distance metric involves divisions and sine computations which cannot be easily supported by homomorphic encryption or secure multi-party computation.

In the next subsection, we introduce the newly designed privacy preserving trajectory clustering which as opposed to what is described in deliverable D3.3 is based on the use of two-party computation (2PC) instead of the additively homomorphic Paillier encryption scheme [34]. The main reason behind the choice of 2PC is because TRACLUS involves a large number of comparisons. As introduced in deliverable D3.1, the original distance metric is approximated with the use of Euclidean distances.

Description. In this section, we describe ppTraclus in details. As previously mentioned, the solution uses two-party computation and distributes the computation among two non-colluding servers S_1 and S_2 (see figure 22). Compared to the original TRACLUS, the distance metric is computed using the Euclidean distance.

By definition, the Euclidean distance (in a two-dimensional space) between two points $P_1(X_1, Y_1)$ and $P_2(X_2, Y_2)$ is computed as defined in Equation 3:

$$ED(P_1, P_2) = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{3}$$

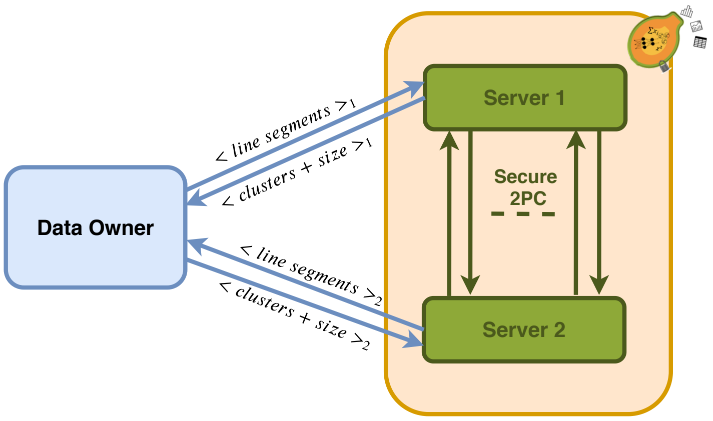

Figure 22: ppTRACLUS

We believe that the perpendicular, parallel and angular distances are partially taken into account when considering the Euclidean distances between each pair of points of the two line segments. Hence, given two line segments, we compute the four Euclidean Distances. Furthermore, since the TRACLUS algorithm uses this distance to compare it with ϵ, we do not need to actually compute the distance but keep the squared Euclidean distances, only. Therefore, given two line segments LS_1 and LS_2, defined with starting points (X_{s1}, Y_{s1}) and (X_{s2}, Y_{s2}) and ending points (X_{e1}, Y_{e1}) and (X_{e2}, Y_{e2}), the distance metric for ppTRACLUS $d_{new}(LS_1, LS_2)$ is defined as follows:

$$\begin{aligned} d_{new}(LS_1, LS_2) = (ED(X_{s1}, X_{s2}))^2 + (ED(X_{s1}, Y_{s2}))^2 + (ED(Y_{s1}, X_{s2}))^2 \\ + (ED(Y_{s1}, Y_{s2}))^2 \end{aligned} \tag{4}$$

In this scenario, the data owner computes two arithmetic shares of the coordinates of the starting and ending points for each line segment. DO sends the first shares to S_1 and the other shares to S_2. The main reason to involve two servers instead of one is to relieve the computation load at DO's side. These two servers further perform the clustering algorithm based on two-party computation and send the partial results to DO. The pseudo code of ppTraclus is depicted in Algorithm 5:

Algorithm 5: ppTraclus for Server S_k

```
Input  : $< \{(X_{si}, Y_{si}); (X_{ei}, Y_{ei})\} >_k$
Output: $l$: number of clusters
set $l = 0$
for each unclustered line segment $LS_i$ defined by $(X_{si}, Y_{si})$ and $(X_{ei}, Y_{ei})$ do
    for all $LS_j$ $(j \neq i)$ do
        compute $< d_{new}(LS_i, LS_j) >_k$
        if $< d_{new}(LS_i, LS_j) >_k > \epsilon_{new}$
        then
        include $LS_j$ in $Neighbors(LS_i)$
    end
    if $size(Neighbors(LS_i)) > MinLns$
    then
    $l = l + 1$
    $C_l = \{LS_i \cup Neighbors(LS_i)\}$
    for all $LS_c \in C_l$ with $c \neq i$ do
        if $size(Neighbors(LS_c)) > MinLns$
        then
        include $LS_c$ in $C_l$
    end
end
```

During the clustering phase, the two servers jointly compute the four squared Euclidean distances between each pair of line segments, sum them to obtain the distance metric of the new ppTRACLUS. This distance is further compared with a certain threshold denoted ϵ_{new} in order to check whether two line segments are neighbors. The number of neighbors of a line segment is computed incrementally and further compared with $MinLns$. If $MinLns$ is reached than this line segment and all its neighbors are regrouped in one cluster. Later on, the algorithm computes the number of neighbors of each neighbor of the actual line segment. If this number again exceeds $MinLns$ then these second-level neighbors also join the actual cluster. Whenever this phase is completed, the servers send the number of clusters and their size to DO. Once DO receives the outputs from both servers, it finally reconstructs the result and obtains the different clusters and their sizes.

3.1.3 Performances

Since ppTRACLUS is an approximated version of TRACLUS and there is no , there is a need for evaluating the consistency of the resulting clusters. The most frequent methodology used for this aim is the silhouette analysis. This methodology consists of the similarity between elements within the cluster and in other clusters. This similarity is evaluated through the **silhouette coefficient (SC)** [35].

By definition, when multiple clusters are defined, the **silhouette coefficient** of a line segment i is defined as:

$$sc(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{5}$$

where $a(i)$ is the mean distance between line segment LS_i and all other elements of its cluster, and, $b(i)$ the smallest mean distance between line segment LS_i and all other line segments that do not share the same cluster with LS_i. The SC value ranges from -1 (poorly clustered) to +1 (well clustered). One disadvantage of the SC score is that it does not take the noise (non-clustered elements) into account. Therefore, the approach in [26] is also considered to add noise into the calculation of SC as a penalty. For a data set with p elements and a noise value n, the final SC score is multiplied with $(p-n)/n$ to obtain SC score with noise penalty, namely SC_{noise}.

Another important method particularly designed to assess the quality of density-based clustering algorithms is the density-based clustering validation (DBCV) index [26]. In contrast to SC, DBCV also takes noise into account in its quality evaluation. Similar to SC score, DBCV index also ranges from -1 to $+1$, where the higher index value indicates a good clustering quality.

In order to find the optimal ϵ' values in our experiments, we use the heuristic defined in [28]. With this heuristic, an optimal epsilon value is determined using the simulated-annealing algorithm [24]: ϵ' is set to the value that minimizes the entropy of the clustering. Therefore, in our experiments in Table 10 and Table 11, all the ϵ' values are selected by considering the same entropy level in simulated annealing.

To compare the validity of ppTRACLUS with the original one, first, we conduct our experiments based on two public data sets in [28], namely the Hurricane data set[5] and the Deer data set[6], using SC, SC_{noise} and DBCV scores as shown in Table 10. Our experimental results show that ppTRACLUS creates more clusters for the Hurricane data set. For instance, in Experiment 1 TRACLUS returns 2 clusters and ppTRACLUS outputs 3 clusters. Additionally, the number of elements marked as noise is larger with ppTRACLUS than with the original TRACLUS. This is particularly because of the approximation of the original distance with our new distance metric in Equation 4. On the other hand, we observe that ppTRACLUS outputs better results compared to TRACLUS with the with respect to SC, SCnoise, and DBCV. The resulting number of clusters for the Deer data set are equal with TRACLUS and ppTRACLUS. Additionally, in Experiment 1, both TRACLUS and ppTRACLUS returns only one cluster. Although SC and DBCV cannot be applied if there is only one resulting cluster this does not necessarily mean that the quality of the clustering algorithm is poor. It simply represents that the algorithm found only one group of similar elements for the given dataset.

Table 11 shows the results of the evaluation of ppTRACLUS and TRACLUS for the Orange's Synthetic Travel data set[7]. We notice the same behaviour with respect to the quality evalua-

[5]The Hurricane data set is a 2-dimensional track data of Atlantic hurricanes from 1950 to 2006 and it consists of 608 trajectories, which corresponds to 18,343 line segments.

[6]The Deer data set corresponds to 2-dimensional movements of deers in 1995. There exist 32 trajectories which corresponds to 20,033 line segments.

[7]Orange's "synthetic" data set (not related to real individuals) is a 2-dimensional trajectory data corresponding to location of people's mobile phones. It has been created and provided by the French telecom operator Orange, based on anonymised indicators of real trajectories. It consists of 40,000 line segments.

Table 10: Clustering quality assessment for TRACLUS and ppTRACLUS on Hurricane and Deer data sets

		Experiment 1		**Experiment 2**	
		TRACLUS	**ppTRACLUS**	**TRACLUS**	**ppTRACLUS**
	(ϵ', minLns)	$(24, 5)$	$(5000, 5)$	$(4, 5)$	$(2250, 5)$
	# of Clusters	2	3	11	13
	Noise	129	150	597	645
	SC	0.27	0.87	0.44	0.79
	SC$_{noise}$	0.27	0.86	0.42	0.76
	DBCV	0.72	0.96649	0.64	0.76
	(ϵ', minLns)	$(400, 3)$	$(1 \times 10^6, 3)$	$(282, 3)$	$(550 \times 10^3, 3)$
	# of Clusters	1	1	2	2
	Noise	1	480	20	1333
	SC	NA	NA	0.089	0.36
	SC$_{noise}$	NA	NA	0.089	0.34
	DBCV	NA	NA	0.47	0.79

Table 11: Clustering quality assessment for TRACLUS and ppTRACLUS on Orange's Synthetic Travel data set

	Experiment 1			**Experiment 2**		
	TRACLUS	**ppTRACLUS**	**ppTRACLUS′**	**TRACLUS**	**ppTRACLUS**	**ppTRACLUS′**
(ϵ', minLns)	$(4200, 3)$	$(450 \times 10^6, 3)$	$(450 \times 10^6, 3)$	$(47 \times 10^3, 3)$	$(13.5 \times 10^9, 3)$	$(13.5 \times 10^9, 3)$
# of Clusters	1	2	48	1	1	2
Noise	796	13092	13506	0	359	361
SC	NA	0.83	0.98	NA	NA	0.82
SC$_{noise}$	NA	0.56	0.65	NA	NA	0.81
DBCV	NA	0.67	0.37	NA	NA	0.98

tion metrics. However, both algorithms output relatively few clusters while privacy-preserving mobility analytics use case may need more precision (i.e., a higher number of clusters). One illustrative example is the identification of typical routes between two locations A and B with respect to the mobility data collected from users without violating their privacy. In this particular example, ppTRACLUS groups all line segments in one cluster because of its expansion factor 5. Therefore, we propose extend ppTRACLUS as ppTRACLUS', which reduces this factor by only taking the first-level neighbors into account to obtain a larger variety of clusters. We run the same two experiments for ppTRACLUS'. Results show that the number of clusters increases with good results in terms of SC, SC$_{noise}$, and DBCV.

As a consequence, ppTRACLUS exhibits similar behaviour like TRACLUS with respect to the number of clusters and even provides better results than TRACLUS when compared to SC, SC$_{noise}$ and DBCV scores. Consequently, with our new distance metric, ppTRACLUS offers efficient privacy protection together with high clustering quality.

3.2 Privacy Preserving Clustering Based on MinHash

In this section, we give the specifications of the privacy-preserving trajectory clustering. The main idea is to encrypt several trajectories, then execute the trajectory clustering so as to obtain, for a set of trajectories, the number of individuals that have used it. Our solution is based on a simplication of the MinHash [7] algorithm which consists in pre-defining the output trajectories.

3.2.1 Specifications

Vector representation of a trajectory. We consider that the studied geographical area has been divided into a set of cells and we define accordingly the set of all possible transitions from one cell to an adjacent one. A transition is denoted by t_i and the universe of all transitions is denoted $\mathcal{U} = \{t_i\}_{i\in[1,n]}$. A trajectory T is then defined by a set of transitions and we say that $t_i \in T$ if the transition is included into T. For each trajectory T_j, we define a vector V$_j$ of size

n as follows:

$$\forall i \in [1, n], \mathsf{V}_j[i] := \begin{cases} 1 \text{ if } t_i \in T_j \\ n+1 \text{ else} \end{cases}$$

MinHash setup. In order to set up the MinHash procedure, we define h distinct permutations of the set $\{1, \cdots, n\}$, each one represented as a vector denoted as Π_j, $j \in [1, h]$. We also need an algorithm, denoted $\mathsf{smallest}$, taking as input a vector V and outputting the smallest element in V.

Characteristic trajectories. We now consider two specific cells (one is called the Origin, and the other one is called the Destination) that we want to study.

We first define ℓ characteristic trajectories that permit to go from the Origin cell to the Destination (in that direction)[8]. We consider that each characteristic trajectory CT is publicly known by its vector representation CV as given above. In our trajectory clustering algorithm, we will consider that each cluster is given by a characteristic trajectory and our purpose is to count the number of element in each cluster.

MinHash algorithm. The next step is to compute the MinHash of a characteristic vector CV.We define the MinHash of a characteristic vector CV as vector of size h such that

$$\mathsf{MinHash}(CV) = (mh_1, \cdots, mh_h) \text{ with } mh_i = \mathsf{smallest}(\Pi_i \times CV) \quad (6)$$

In this equation 6, $\Pi_i \times CV$ refers to the product of two vectors componentwise.

3.2.2 Encryption scheme and procedures

We know details the encryption scheme and the associated procedures that are used for the computation of privacy preserving clustering en the encrypted domain. We use a 2 Party Computation (2PC) scheme based additive secret sharing. For the procedures needing comparison we use the algorithm from [12]. Following, the list of the 2PC procedures:

2PCInit Initialize the 2PC protocols.

2PCEncrypt Takes a message m and returns the corresponding share c.

2PCDecrypt Gather the shares and returns the message m.

2PCMult Takes two ciphertexts $c_0 = \mathsf{2PCEncrypt}(m_0)$, $c_1 = \mathsf{2PCEncrypt}(m_1)$, and return a new ciphertext $c = \mathsf{2PCEncrypt}(m_0 \cdot m_1)$.

2PCAdd Takes two ciphertexts $c_0 = \mathsf{2PCEncrypt}(m_0)$, $c_1 = \mathsf{2PCEncrypt}(m_1)$, and return a new ciphertext $c = \mathsf{2PCEncrypt}(m_0 + m_1)$.

[8]There are different ways to automatically create those characteristic trajectories and this is not the purpose of this document to give them. We just consider that this step has already been done, using any existing method.

2PCScalarMult Takes one ciphertext $c_0 = \text{2PCEncrypt}(m_0)$ and a scalar a, then returns a new ciphertext $c = \text{2PCEncrypt}(a \cdot m_0)$.

2PCScalarAdd Takes one ciphertext $c_0 = \text{2PCEncrypt}(m_0)$ and a scalar a, then returns a new ciphertext $c = \text{2PCEncrypt}(a + m_0)$.

2PCLesser Takes two ciphertexts $c_0 = \text{2PCEncrypt}(m_0)$, $c_1 = \text{2PCEncrypt}(m_1)$, and return a new ciphertext encrypting 1 if $m_0 \leq m_1$, 0 otherwise.

2PCMin Takes a set of ciphertexts $\{c_i = \text{2PCEncrypt}(m_i)\}_{i \in [1,h]}$, and returns a new ciphertext $c = \text{2PCEncrypt}(\min(m_1, \cdots, m_h))$.

2PCEqual Takes two ciphertexts $c_0 = \text{2PCEncrypt}(m_0)$, $c_1 = \text{2PCEncrypt}(m_1)$, and return a new ciphertext encrypting 1 if $\{m_0 = m_1\}$, 0 otherwise

The above procedures can be apply directly to vectors, by applying the procedures to the vector's elements independently. In the following section we don't distinguish when a procedure is called on a vector of ciphertexts or a single ciphertext. We described the necessary procedures, let see how to applies them to performs MinHash trajectory Clustering in the encrypted domain.

3.2.2.1 MinHash trajectory clustering in the encrypted domain

The next step is to create trajectory clusters from a set of trajectories. We consider then a set of N trajectories represented as N vectors V_i, $i \in [1, N]$, as defined above. There are four steps in such process:

1. the encryption of the trajectories;
2. the computation of the MinHash of each trajectory;
3. the creation of the clusters;
4. the decryption of the number of elements in each cluster.

We now give some details about each step.

Trajectory encryption. Given a vector V representing a trajectory, the encryption of V simply consists in encrypting each component of the vector, as:

$$\forall k \in [1, n],\ \mathsf{EV}[k] = \text{2PCEncrypt}(\mathsf{V}[k], \mathsf{pk}).$$

The output ciphertext is then the vector EV.

MinHash computation. The idea is to make use of the MinHash algorithm given above, but now defined in the encrypted domain. More precisely, we consider that a trajectory is close to another one if the MinHash of both trajectories are equal[9]. For that purpose, we make use of the basic procedure related to the manipulation of ciphertexts, given in Section 3.2.2.

1. $\forall j \in [1, h]$, generate the vector $\mathsf{EMH_j} = [\mathsf{2PCScalarMult}(\mathsf{EV}[k], \Pi_j[k])]_{k\in[1,n]}$;
2. $\mathsf{EncMinHash}(\mathsf{EV}) = [\mathsf{2PCMin}(\mathsf{EMH}_j)_{j\in[1,h]}]$

Clusters creation. Having computed the MinHash of all the vectors representing the trajectories, the next step consists in creating the clusters. In fact, in our solution, each cluster is related to each characteristic trajectory that has been set up previously: there are then ℓ clusters in our trajectory clustering method.

The cluster creation is then reduced to the problem of computing the number of trajectories that are closed to those characteristic ones. This is done by comparing the computed MinHash of each trajectory with the ones of the characteristic trajectories, which can be done as follows in the encrypted world.

1. $\forall i \in [1, \ell]$, $\mathsf{ecount}[i] = \mathsf{2PCEncrypt}(0)$;
2. $\forall i \in [1, \ell], \forall k \in [1, N]$,

$$\mathsf{ecount}[i] = \mathsf{2PCAdd}(\mathsf{ecount}[i], \mathsf{2PCEqual}(\mathsf{MinHash}(\mathsf{CV}_i), \mathsf{MinHash}(\mathsf{EV})))$$

At the end of this process, each counter $\mathsf{ecount}[i]$ contains the number of items in the cluster i represented by the characteristic trajectory CT_i.

Clusters decryption. The final step consists at permitting the decryption of the result, but if and only if the counter is greater than a defined upper bound, denoted κ, so as to obtain the κ-anonymity. Otherwise, the result is defined as 0. We here make use of a two-party computation, which is executed ℓ times, for each counter to be able to compare each counter.

1. $c_{\leq}[i] = \mathsf{2PCLesser}(\mathsf{ecount}[i], \kappa)$
2. $c[i] = \mathsf{2PCMult}(c_{\leq}[i], \mathsf{ecount}[i])$
3. $\mathsf{count}[i] = \mathsf{2PCDecrypt}(c[i])$

In the end we obtain a set $\{\mathsf{count}[i]\}_{i\in[1,l]}$ such that $\mathsf{count}[i]$ is the number of trajectories in the cluster CT_i if there are more than κ trajectories.

[9]A better solution could be to make use of similarities, and then define a more refined method which compute a value between 0 and 1, and then conclude that two trajectories are close iff such value is more than some predefined upper bound. This may be done in a second version of the specifications.

3.2.3 Implementation and performances

We implemented this solution using Facebook CrypTen library [1] which provides the 2PC protocol and the procedures described in Subsection 3.2.2. We run some benchmarks on a different set of parameters to estimate the performances that represent different scales of the use case described in D2.1. The first parameter to benchmark is the number of transitions in a trajectory, the more a trajectory contains transitions the more precise it is. The trajectory size ranges from 50 to 500, see table 12. The second parameter to benchmark is the number of permutations used in the MinHash algorithm, the more permutations used the more refines are the clusters computed. The number of permutations ranges from 10 to 150, see table 12.

In the computation, the most expensive operation is 2PCMin. The min operation complexity is quadratic in the number of transitions in a trajectory, and we have to repeat the operation for each permutation.

In practice, in the use case we are interested in, we use 150 transitions per trajectory and 100 permutations.

Table 12: Computation time for the clustering of 100 trajectories in 10 clusters.

# transitions	# permutations	Time (s)
50	10	1.22
	25	3.31
	50	6.38
	100	11.90
	150	18.72
100	10	4.86
	25	11.60
	50	22.78
	100	47.26
	150	68.36
150	10	13.82
	25	34.40
	50	68.76
	100	137.43
	150	206.55
200	10	65.35
	25	161.52
	50	322.50
	100	643.70
	150	961.12

4 Privacy Preserving Counting

In this section, we give the specifications of the privacy-preserving counting primitive based on Bloom Filters. The main idea is to encrypt one or several sets of data related to some individuals, and optionally perform some basic operations on them (such as union or intersection) and finally count the number of individuals in the set. Remind that in the use case we are interested in counting up to 1 million people each 30 min. Unless specify otherwise, in this section $[a, b]$ means $[a, b] \cap \mathbb{Z}$. The $||$ operator stands for concatenation.

4.1 Specifications

We first start by describing how bloom filters works, then the cryptographic protocol used for the solution. Finally privacy preserving counting is presented.

4.1.1 Bloom Filters

Let $\mathcal{D} = \{d_1, \cdots, d_i, \cdots, d_\ell\}$ be a finite set of some entries denoted by d_i. As the size ℓ of the set can be very large, we assume that the set is represented by a counting table denoted $\mathcal{T}$. The option that we consider is to make use of Bloom filters.

Bloom filters creation. A Bloom filter is used to store a set of data of variable size into a bit-string of fixed size. It is then possible, with a simple test, to estimate the probability that an element is in the set of data (which depends on the size of the resulting bit-string and on the number of data). This probability is equal to 0 if the output of the test is “no”. More precisely, the result is an m-bit string named BF. We note BF $=$ BF$[m-1] \cdots$ BF$[1]$BF$[0]$ where BF$[i]$ denotes the $i-$th bit of BF, and each BF$[i] \in \{0, 1\}$. Initially, BF$[i] = 0$ for all $i \in [0, m-1]$. Let us define q hash functions $\mathcal{H}_1, \cdots, \mathcal{H}_q$ where each $\mathcal{H}_k : \{0,1\}^* \longrightarrow \{0,1\}^\gamma$ with $m = 2^\gamma$. For $i \in [1, \ell]$, the Bloom filter creation consists of computing $\mathcal{H}_1(d_i), \cdots, \mathcal{H}_q(d_i)$. For every $k \in [1, q]$, we assign BF$[l] = 1$ where $l = \mathcal{H}_k(d_j)$.

Let BF be a Bloom filter and let d be a new data to be put in this Bloom filter. We define the `update` function as

$$\text{BF} \quad = \quad \mathtt{update}(\text{BF}, d)$$

which assigns BF$[\mathcal{H}_k(d)] = 1$ for $k \in [1, q]$.

Bloom filters testing. If one wants to know the element $\tilde{d}$ is in the set $\mathcal{D}$ by using the created Bloom filter, one has to compute $\tilde{l}_k = \mathcal{H}_k(\tilde{d})$ for every $k \in [1, q]$. If there is an element $k_0 \in [1, q]$ such that B$[\tilde{l}_{k_0}] = 0$ then, $\tilde{d} \notin \mathcal{D}$, otherwise, $\tilde{d} \in \mathcal{D}$. Such method is probabilistic in the sense that if the above testing procedure outputs that $\tilde{d} \in \mathcal{D}$, with an error probability of about $\left(1 - e^{\frac{-\ell q}{m}}\right)^q$.

Approximating the number of entries. Let us consider a Bloom filter BF of length m, that has been filled in using q hash functions for the set $\mathcal{D}$. Let n be the number of 1 in BF (which can be computed as $n = \sum_{i=0}^{m-1} \mathsf{BF}[i]$). It exists a formula [39] permitting to approximate the number of items $\tilde{\ell}$ in the set $\mathcal{D}$. More precisely, this is given by

$$\tilde{\ell} \quad = \quad -\frac{m}{q} \ln\left(1 - \frac{n}{m}\right).$$

In our use case, we need to provide κ-anonymity which necessitates that the number ℓ of items in the Bloom filter can only be known (in our case decrypted, see below) if $\ell \geq \kappa$. From the above formula, we can derive the maximum number n_{max} of bits that need to be set to 1 in the Bloom filter BF to reach such κ-anonymity, as:

$$n_{max} \quad = \quad m \cdot \left(1 - e^{-\frac{\kappa q}{m}}\right).$$

In the sequel, we consider that those values κ and n_{max} are known and public.

Bloom filters union. The size $\ell_\cup$ of the union of two sets $\mathcal{D}_0$ and $\mathcal{D}_1$, represented by two Bloom filters BF_0 and BF_1 (defined again with size m and q hash functions) can be estimated by the following formula:

$$\tilde{\ell_\cup} \quad = \quad -\frac{m}{k} \ln\left(1 - \frac{n_\cup}{m}\right)$$

where $n_\cup$ is the number of bits set to one in either of (or both) the two Bloom filters.

In the encrypted domain, the above strategy can easily be derived to the case of two Bloom filters. To compute $\tilde{\ell_\cup}$, it remains to compute $n_\cup$, which can be done using the appropriate homomorphic encryption which permits to compute $\mathsf{BF} \cup \widetilde{\mathsf{BF}}$ (OR operation). In fact, $\mathsf{BF} \cup \widetilde{\mathsf{BF}}$ can be computed as

$$\mathsf{BF} \cup \widetilde{\mathsf{BF}} \quad = \quad \mathsf{BF} + \widetilde{\mathsf{BF}} - \mathsf{BF} \times \widetilde{\mathsf{BF}} \ (\text{in } \mathbb{Z}),$$

which gives us a way to compute the encrypted Bloom filter corresponding to the union of two encrypted Bloom filters. We will then simply have to make use of the approximation of the number of entries in this resulting Bloom filter to reach our objective.

Bloom filters intersection. Let $\mathsf{BF}_0, \mathsf{BF}_1$ be two Bloom filters. Let l_0, l_1 be the estimation of the number of elements in $\mathsf{BF}_0, \mathsf{BF}_1$ respectively. Let $l_\cup$ the estimation of the number of elements of the union of $\mathsf{BF}_0, \mathsf{BF}_1$. The intersection of two Bloom filters $\mathsf{BF}_0, \mathsf{BF}_1$ can be estimated as

$$\tilde{\ell_\cap} = l_0 + l_1 - l_\cup,$$

which can easily be solved using the above results.

4.1.2 Overall Architecture

We consider the overall architecture, composed of the Orange interface which collects the probe data, the Platform ClientSideAgent which prepares the data to be treated in the encrypted domain, the PP data analytics platform which performs the analytics in the encrypted domain, and finally the Requestor Side Agent which is the requestor of the analytics, and which is involved in the decryption part. The overall interactions are given in Figure 23.

We consider one spatio-temporal window (the whole protocol can be repeated for each new spatio-temporal window) with n indicators. Each indicator $j \in [1, n]$ is related to one unique Bloom filter BF_j.

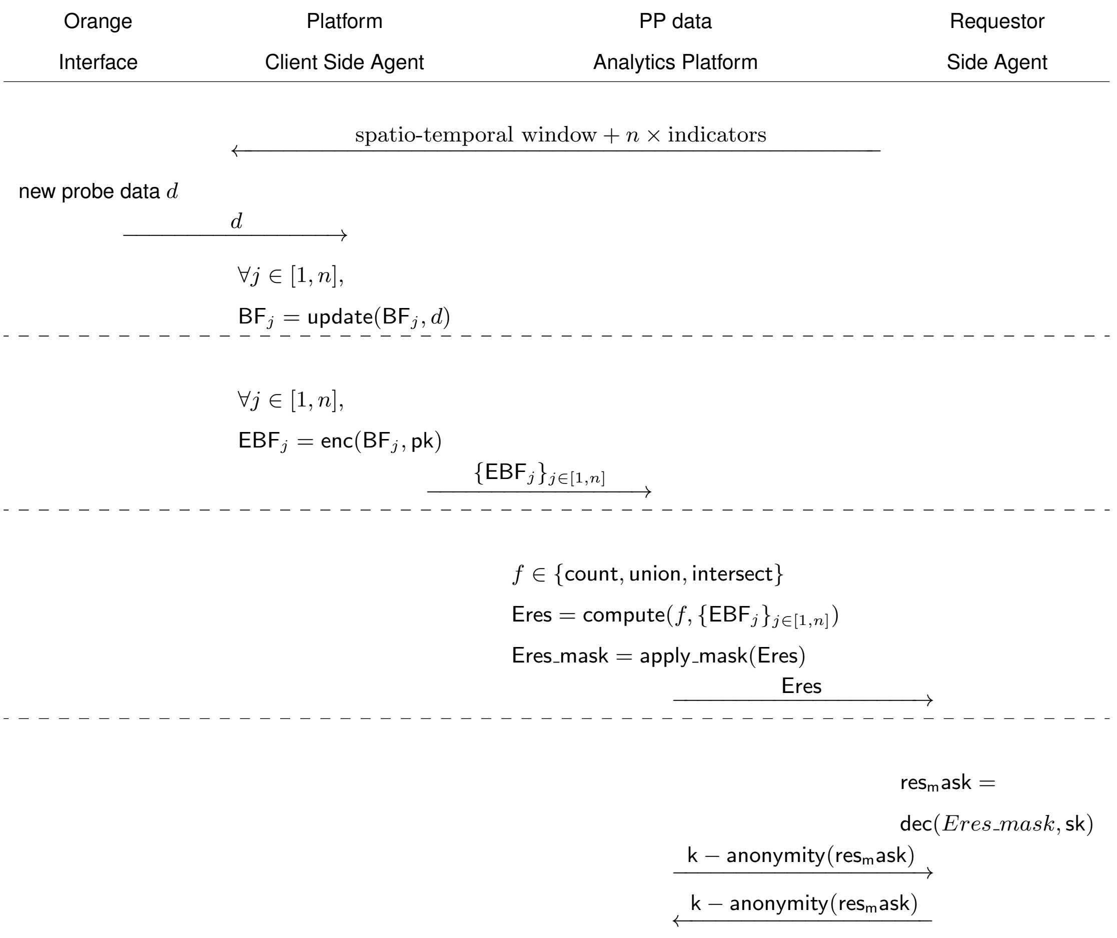
Orange Interface
Platform Client Side Agent
PP data Analytics Platform
Requestor Side Agent
spatio-temporal window $+ n \times$ indicators
new probe data d
d
$\forall j \in [1, n]$,
$\mathsf{BF}_j = \mathsf{update}(\mathsf{BF}_j, d)$
$\forall j \in [1, n]$,
$\mathsf{EBF}_j = \mathsf{enc}(\mathsf{BF}_j, \mathsf{pk})$
$\{\mathsf{EBF}_j\}_{j \in [1,n]}$
$f \in \{\mathsf{count}, \mathsf{union}, \mathsf{intersect}\}$
$\mathsf{Eres} = \mathsf{compute}(f, \{\mathsf{EBF}_j\}_{j \in [1,n]})$
$\mathsf{Eres\_mask} = \mathsf{apply\_mask}(\mathsf{Eres})$
Eres
$\mathsf{res_m ask} =$
$\mathsf{dec}(Eres\_mask, \mathsf{sk})$
$\mathsf{k - anonymity(res_m ask)}$
$\mathsf{k - anonymity(res_m ask)}$

Figure 23: Overall architecture

4.1.3 Encryption scheme and procedures

We now detail the cryptographic parts and the used encryption scheme, together with the different basic procedures that will be useful to count and make the union and the intersection of Bloom filters. For this case we are using a mix of Homomorphic Encryption (HE) and 2 Party Computation (2PC). We use the BFV[16] scheme for homomorphic encryption and Additive Secret Sharing for 2 party computation. We use the packing property of the BFV scheme to allow us to encrypt multiple bits of a bloom filter into one ciphertext. However, with the BFV scheme the secure comparison to perform the k-anonymity check is very expensive. To tackle this problem we are using 2PC and the secure comparison algorithm used in [12].

Homomorphic Encryption procedures. The BFV provides the following set of procedures that we use in the for the privacy preserving counting:

HEKeyGen generates the followings keys:

- **sk** the secret key used for the decryption.
- **pk** the public key used for encryption.
- **rt** the rotation key used to performs the rotation procedure.

HEEncrypt encrypts some data m using the public key pk, returning a ciphertext c.

HEDecrypt decrypt a ciphertext c using the secret key sk, returning data m.

HEAdd takes two ciphertexts $c_0 = \text{HEEncrypt}(m_0)$, $c_1 = \text{HEEncrypt}(m_1)$, and return a new ciphertext $c = \text{HEEncrypt}(m_0 + m_1)$.

HEMult takes two ciphertexts $c_0 = \text{HEEncrypt}(m_0)$, $c_1 = \text{HEEncrypt}(m_1)$, and return a new ciphertext $c = \text{HEEncrypt}(m_0 \cdot m_1)$.

HEScalarMult takes one ciphertext $c_0 = \text{HEEncrypt}(m_0)$ and a scalar a, then returns a new ciphertext $c = \text{HEEncrypt}(a\dot{m}_0)$.

HEScalarAdd takes one ciphertext $c_0 = \text{HEEncrypt}(m_0)$ and a scalar a, then returns a new ciphertext $c = \text{HEEncrypt}(a + m_0)$.

HERotate takes one ciphertext $c_0 = \text{HEEncrypt}(m_0)$, a stride s, and the rotation key rk, then returns a new ciphertext c where the slots has been rotated by a stride s.

2PC procedures. Here the procedures provided by the 2PC scheme that will be used:

2PCInit initialize the 2PC protocols.

2PCEncrypt takes a message m and returns the corresponding share c.

2PCDecrypt gathers the shares and returns the message m.

2PCSub takes two ciphertexts $c_0 = \text{2PCEncrypt}(m_0)$, $c_1 = \text{2PCEncrypt}(m_1)$, and return a new ciphertext $c = \text{2PCEncrypt}(m_0 - m_1)$.

2PCScalarAdd takes one ciphertext $c_0 = \text{2PCEncrypt}(m_0)$ and a scalar a, then returns a new ciphertext $c = \text{2PCEncrypt}(a + m_0)$.

2PCLesser takes two ciphertexts $c_0 = \text{2PCEncrypt}(m_0)$, $c_1 = \text{2PCEncrypt}(m_1)$, and return a new ciphertext encrypting 1 if $m_0 \leq m_1$, 0 otherwise.

4.1.4 Privacy preserving counting

Now that we described the encryptions schemes used for this PETS, let us see how we use them to implement a privacy preserving counting service. We first describe the bloom filter encryption procedure. Then we see how to count the elements to estimate the cardinality of the bloom filter while preserving the κ-anonymity property. Finally, we give the procedures to estimate the cardinality of the union and the intersection of two bloom filters.

Bloom filter encryption. Let BF be a bloom filter of n bits, we will denote EBF the encryption of BF. In most application the number of bits used in a bloom filter is far greater that the number of slots available in a single ciphertext. So, in practice the bloom filter is split into parts that fit in a ciphertext. Let m be the number of slot in a ciphertext. We split a bloom filter BF into $k = n/m$ parts,

$$\begin{aligned} \mathsf{BF} &= b_0 b_1 ... b_{n-1} b_n \\ &= b_1 b_2 \cdots b_{m-1} b_m || \cdots || b_{(k-1)m+1} b_{(k-1)m+2} \cdots b_{km-1} b_{km} \end{aligned}$$

The encryption of BF is a set of ciphertexts as following

$$\mathsf{EBF} = \{\mathsf{HEEnc}(b_{im+1} b_{im+2} \cdots b_{(i+1)m-1} b_{im})\}_{i \in [0,k]}$$

Bloom filter sum. To estimate the cardinality of a bloom filter we first need to count the number of 1 in it. Hence, in our case we need to do the sum of each slot in a ciphertext. To do that, we use the rotation property of the BFV scheme that allow us to permute the slots in a ciphertext. We modified the TotalSum algorithm from [18] to fit our case see Algorithm . The main idea of the algorithm is to sum the ciphertext with its rotation by a power of two. At the end, the first slot of the resulting ciphertext contains the sum of the slots.

Algorithm 6: SumCiphertext

```
Input  : $C$ a ciphertext with $m$ slots.
Output: $C_{out}$: a ciphertext where the first slot contain the sum of all the slots.
$C_{out} = C$
for $i$ from 0 to $\log_2(m) - 2$ do
    $C_{rot} = \mathsf{HERotate}(C, 2^i, \mathsf{rk})$
    $C_{out} = \mathsf{HEAdd}(C_{out}, C_{rot})$
end
return $C_{out}$
```

Now that we know how to sum the slots of one ciphertext we can sum an entire encrypted bloom filter

Algorithm 7: BFSum

```
Input  : EBF = $\{\text{HEEnc}(b_{im+1}1b_{im+2}\cdots b_{(i+1)m-1}b_{im})\}_{i\in[0,k]}$ an encrypted
         ciphertext.
Output: $C_{out}$: a ciphertext where the first slot contain the sum of all the bloom filter.
$C_{out} = HEEncrypt(0, \text{pk})$
for $i$ from 0 to $\kappa$ do
    $C_{out} = \text{HEAdd}(C_{out}, \text{SumCiphertext}(C_{rot}))$
end
return $C_{out}$
```

At this step, we could send back the result to the requestor and estimate the cardinality. But, first we need to ensure the κ-anonymity of the result.

κ-anonymity. To check the κ-anonymity, we need to compare if the sum we just computed is greater than $k \cdot q$ where q is the number of hash functions used to compute the bloom filter see Subsection 4.1.1. Has stated above, doing the comparison directly with homomorphic encryption leads to very performances. So, we choose to switch to a 2PC protocol to compute the comparison.

The main issue is how to switch encryption scheme while preserving the security of the data. Indeed, to decrypt the ciphertext we need to send it back to the requestor, who can find the result. To tackle this problem, before sending back the ciphertext, the platform generates a random number greater than $k \cdot q$ and adds it to the ciphertext. Doing so will prevent the requestor to recover the orignal value during the decryption. We can then proceed with the comparison using 2PC. If the κ-anonymity check fails, the requestor get -1 as a result. Algorithm 8 describes the protocol for the platform. Algorithm 9 describes the protocol for the requestor.

Algorithm 8: Platform k-anonymity

```
Input  : $C$ a ciphertext containing the sum of the bloom filter. $q$ the number of hash
         function.
Output:
$r = \text{random}()$
$C = \text{HEScalarAdd}(C, r)$
Send $C$ to requestor.

$\text{2PCInit}()$
$C_r = \text{2PCEncrypt}(r \cdot k \cdot q)$
$\tilde{C} =$ Receive share for $C$ from requestor
$C_b = \text{2PCLesser}(C_r, \tilde{C})$
$\tilde{C} = C_b \cdot -1 + (1 - C_b) \cdot \tilde{C}$
Send $\tilde{C}$ to requestor.
```

Algorithm 9: Requestor k-anonymity

```
Input  : $C$ a ciphertext containing the sum of the bloom filter. $q$ the number of hash
         function.
Output: $res$ the sum of the bloom filter if $res \geq k$.
Receive $C$ from platform.
$tmp = \text{HEDecrypt}(C, sk)$

$\text{2PCInit}()$
$\tilde{C} = \text{2PCEncrypt}(r)$ $C_r =$ Receive share for $C_r$ from platform
$C_b = \text{2PCLesser}(C_r, \tilde{C})$
$\tilde{C} = C_b \cdot -1 + (1 - C_b) \cdot \tilde{C}$
$res = \text{2PCDecrypt}(\tilde{C})$
if $res \neq -1$: compute cardinality using formula from 4.1.1.
```

We described our protcol to estimate the cardinality of a bloom filter while ensuring the privacy of the data. Now let us have a look at the union and intersection of two bloom filters.

4.1.5 Union of Bloom Filters

The next step corresponds to the way we can compute the number of entries in the union of two or more Bloom filters. For this purpose, we make use of the technique given in Section 4.1.1. In this section, we only describe the case of two Bloom filters. The generalization to more can easily be performed.

More precisely, we describe the $\text{union}(\text{EBF}, \widetilde{\text{EBF}})$ procedure, which takes as input two encrypted Bloom filters EBF and $\widetilde{\text{EBF}}$. The whole protocol is given in Algorithm 10 and is split into two parts: the computation, in the encrypted domain, of the Bloom filter corresponding to $\text{BF} \cup \widetilde{\text{BF}}$, and then the computation of number of individuals in the resulting filter, in a privacy-preserving way.

Algorithm 10: UnionBF.

```
Input  : EBF, $\widetilde{\mathsf{EBF}}$ two encrypted bloom filters.
Output: $C_\cup$: a ciphertext of the union of EBF, $\widetilde{\mathsf{EBF}}$.
$C_\times = \mathsf{HEMult}(\mathsf{EBF}, \widetilde{\mathsf{EBF}})$
$C_1 = \mathsf{HEScalarMult}(C_\times, -1)$
$C_\cup = \mathsf{HEAdd}(C_1, \mathsf{EBF}, \widetilde{\mathsf{EBF}})$
return BloomFilterCount($C_\cup$)
```

4.1.6 Intersection of Bloom Filters

We finally show how one can compute the number of entries in the intersection of two or more Bloom filters. Using the simple formula given in Section 4.1.1, we define the $\mathtt{intersect}(\mathsf{EBF}, \widetilde{\mathsf{EBF}})$ procedure which takes as input two encrypted Bloom filters EBF and $\widetilde{\mathsf{EBF}}$ and output the result. The whole protocol is given in Algorithm 11.

Algorithm 11: Secure computation of the intersection of two bloom filters.

```
Input  : EBF, $\widetilde{\mathsf{EBF}}$ two encrypted bloom filters.
Output: $C_\cup$: a ciphertext of the union of EBF, $\widetilde{\mathsf{EBF}}$.
$C$ =BFSum($EBF$)
$\tilde{C}$ =BFSum($\widetilde{EBF}$)
$C_\times = \mathsf{HEMult}(\mathsf{EBF}, \widetilde{\mathsf{EBF}})$
$C_1 = \mathsf{HEScalarMult}(C_\times, -1)$
$C_\cup = \mathsf{HEAdd}(C_1, \mathsf{EBF}, \widetilde{\mathsf{EBF}})$
$C_1 = \mathsf{HEScalarMult}(C_\cup, -1)$
$C_{res} = \mathsf{HEAdd}(C, \tilde{C}, C_{res})$
return BloomFilterCount($C_{res}$)
```

4.2 Implementation and Performances

In the previous subsection we detailed the specifications of the privacy preserving counting using bloom filters. In this subsection we describe the implementation and the performances for different scenarii.

For the implementation we use Microsoft SEAL library[10] for the BFV scheme, and Facebook Crypten [1] library for the 2PC protocol and the procedures described in Section 4.1.3.

To evaluate the performance of our solution, we generate bloom filters with 7 hash functions and a probability of false positive of 1%. We chose parameters differents size of bloom filter according to the number of elements we want to count (see Table 13).

In the Table 14 one can find the performances for the three operations:

1. estimating the cardinality of one bloom filter,

[10]Simple Encrypted Arithmetic Library (SEAL): https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/

Table 13: Size of a bloom filter depending on the number of elements to count

# elements to count	Bloom Filter Size
100	960
1 000	9 593
100 000	959 296
1 000 000	9 592 955

2. estimating the cardinality of the union of two bloom filters,
3. estimating the cardinality of the intersection of two bloom filters.

In Table 14, we provide the performance of our solution depending on the number of elements we have to count. Remind that in the corresponding use case described in D2.2, we want to be able to count up to 1 million people in a 30 minutes window. We can see that our solution fits the requirements as all the operations takes less than a minute.

Table 14: Performance of the different operations depending on the number of elements to count.

# elements to count	Operation	Time (s)
100	card	0.118
	card union	0.323
	card inter	0.845
1000	card	0.162
	card union	0.70
	card inter	1.33
100000	card	0.604
	card union	4.30
	card inter	6.09
1000000	card	6.1
	card union	40.76
	card inter	54.38

5 Privacy Preserving Statistics

Unless specify otherwise, in this section $[a, b]$ means $[a, b] \cap \mathbb{Z}$.

5.1 From Homomorphic Encryption

Main idea. Each user $\mathcal{U}_i$ for $i \in \mathcal{I}$ has some input x_i. The requestor $\mathcal{R}$ wants to obtain some statistics over the x_i's. For example, this corresponds to an aggregation of a subset of the inputs, i.e $\sum_{i\in\mathcal{I}'} x_i$ for a possible $\mathcal{I}' \subseteq \mathcal{I}$. The naive solution is to send each x_i (in the clear) to a (trusted) executor $\mathcal{E}$ that will aggregates all the values and send the final result to $\mathcal{R}$ which does not learn more information if there is no collusion with the executor.

If $\mathcal{E}$ or $\mathcal{R}$ are not trusted, each user $\mathcal{U}_i$ wants to ensure that its input x_i is not learned during the execution of the protocol. In particular, we want to ensure the **privacy** of each individual regarding their input x_i.

A first attempt is to use an encryption mechanism that is sufficiently malleable in order to do the computations. In particular, supposes that a requestor $\mathcal{R}$ has a public key (resp. a secret key) for a linearly homomorphic encryption (LHE) scheme denoted by $\mathsf{pk}_{\mathcal{R}}$ (resp. $\mathsf{sk}_{\mathcal{R}}$). The protocol is as follows:

- Each $\mathcal{U}_i$ sends to $\mathcal{E}$ an encryption of its input x_i using the LHE scheme with the public key $\mathsf{pk}_{\mathcal{R}}$, i.e LHE.Encrypt$(x_i, \mathsf{pk}_{\mathcal{R}})$.
- The executor $\mathcal{E}$ can evaluate a linear function over all the received ciphertexts to obtain a ciphertext C that decrypt to a $\sum_{i\in\mathcal{I}'} \lambda_i x_i$ for every possible $\mathcal{I}' \subseteq \mathcal{I}$ and λ_i. It sends the result to the requestor $\mathcal{R}$.
- The requestor $\mathcal{R}$ uses its corresponding secret key $\mathsf{sk}_{\mathcal{R}}$ in order to obtain the final result which corresponds to a linear function of the underlying data.

It is possible to split this protocol in two stages. The first stage corresponds to an accumulation process, where the executor obtain all the ciphertexts for the users that participate to the protocol. Then in the second stage, the computation is done. Notice that it is also possible to use a fully homomorphic encryption scheme and evaluate any function over the data. We next overview some issues with this basic protocol.

1. Each user can *cheat* and sends a fake element $\tilde{x}_i$.
2. Since the final result is obtained after the computation, the executor $\mathcal{E}$ could try to obtain more information by sending for example only a ciphertext corresponding to some specific x_i or any subset of the users inputs. In particular, it could learn any partial sum.
3. If $\mathcal{R}$ learns one of the ciphertext (for example by colluding with $\mathcal{E}$), then it could obtain the underlying x_i.

A way to solve this problem is to use Non-Interactive Zero-Knowledge (NIZK) proofs [6] in order to convince the other parties that the computations were done correctly. In addition, we have to suppose in this situation that $\mathcal{E}$ and $\mathcal{R}$ does not collude.

5.2 From Secret Sharing

Another solution is possible by using (threshold) secret-sharing schemes. Recall that secret-sharing gives the capability of splitting some secret input into different shares. Then, it is possible from a (threshold) number of shares to reconstruct the original secret input.

Based on the structure of some secret sharing schemes, there exists some properties that are useful. For example, using Shamir Secret Sharing [36], it is possible to obtain from shares of two inputs, the sum of the original inputs by just adding the shares. This idea is used for the sum function by Prio (Ref to be added).

Each user $\mathcal{U}_i$ will share its own input x_i as two inputs $([x_i]_\mathcal{E}, [x_i]_\mathcal{R})$ such that $x_i := [x_i]_\mathcal{E} + [x_i]_\mathcal{R}$. The main idea of Prio is to send $[x_i]_\mathcal{E}$ to $\mathcal{E}$ and $[x_i]_\mathcal{R}$ to $\mathcal{R}$ and remark that

$$\sum_{i\in\mathcal{I}} x_i := \sum_{i\in\mathcal{I}}([x_i]_\mathcal{E} + [x_i]_\mathcal{R}) = \sum_{i\in\mathcal{I}}[x_i]_\mathcal{E} + \sum_{i\in\mathcal{I}}[x_i]_\mathcal{R}.$$

The two last sums could be computed by each $\mathcal{E}$ and $\mathcal{R}$ individually.

Regarding the above discussed problems of the previous solution: the user could still fake its input; the executor $\mathcal{E}$ (or the requestor $\mathcal{R}$) has to provide the computation over the shares, otherwise it is not possible to reconstruct the final result (the sum) from the shares; and the collusion condition is inevitable, since otherwise, it is possible to obtain the original x_i from only two shares of the same user.

Considering NIZK, Prio proposes only a user's corruption (using techniques from MPC) with less expensive solution (compared to NIZK). In their work, the user has to produce a valid proof of the input's format to convince the executor and the requestor about the validity of the shares. Notice that the paper assumes some trust between $\mathcal{E}$ and $\mathcal{R}$.

5.3 From (variants of) Multi-Client FE

In (decentralized) multi-client FE (DMCFE) [11] a group $\mathcal{I}$ of users has some secret keys sk_i for all $i \in \mathcal{I}$ with two capabilities:

1. in the encryption procedure, each user can encrypt its input x_i using some predetermined label; and
2. in a *functional key generation* procedure, the users could interactively or not participate to generate a functional key sk_f.

A decryptor which has access to all the ciphertexts and a functional key for f, reconstructed from *all* users in this group under the same label, could obtain in clear the evaluation $f(\{x_i\}_{i\in\mathcal{I}})$.

When implementing this primitive, the Setup procedure (generating the sk_i and other parameters) needs somehow to be interactive between all participants in the protocol. In DMCFE, this interaction is only fixed once for all during this Setup phase and all the other algorithms are non-interactive.

At first glance, it suitable to use DMCFE for our problem: the users dictate the nature of the computation. In particular, it is not possible to obtain other than the predetermined function whenever we have the functional keys and the ciphertexts. Note however that it does not give a complete answer to our problem.

1. Each user has to interact in advance with the other participants in the Setup phase. The set of users is then fixed and if one user does not contribute with (its part of) a *functional key*, it is not possible (by definition) to obtain the evaluated function over the remaining users. In our case, there is some users that will not participate, so this solution (DMCFE) is not *fault-tolerant*.

2. The roles of $\mathcal{E}, \mathcal{R}$ are not clear in this context. In general, DMCFE is more suitable where there is a set of users and a one decryptor (for example $\mathcal{R}$).

Solving item 1. In a recent work, Chotard et al. introduce the notion of *dynamic* decentralized MCFE which try (among other concerns) to solve the item 1. Roughly speaking, the users are sorted in groups (or lists), and each user can decide to contribute in the protocol for any subset of users of its choice. The evaluation of the data is revealed only when all parties in the *same* group have sent their contributions.

The authors claim that their solution for the inner product functionality is dynamic since each user could generate its own public/secret keys and join the group without any interaction of the others. However, even if it gives a huge flexibility and the definition seems to cover potentially many use-cases, it does not consider any users failure in their construction[11].

- Indeed, the users needs, as in the normal DMCFE, to know in advance the group of users for which they are trying to collaborate. In particular, if a user does not contribute with a functional key contribution, then again by definition it is not possible to obtain the final evaluation.

- On a *real-world* application of this primitive, the users could have sent their contributions for nothing if only one user contribution is missing.

In an other related work, the Ad-hoc multi-input FE notion is a functional encryption where users can join (by encrypting input) the system on-the-fly, and functional keys can be generated in a decentralized way, by each client, without any interaction. The main differences with Dynamic DMCFE is that the Setup phase does not need to consider labels.Moreover, since each user encrypts its input once for all using some public parameters, the work is transferred into the functional key generation procedure where, when asked, the user needs to know the set of users for which the functional key is issued. Indeed, the aggregator could accumulate any choice of ciphertexts and it would eventually ask the different involved parties to send the partial decryption keys corresponding to this ciphertext.

Notice that in particular, if one user fails to send one partial decryption key, then it is not possible to properly decrypt. This leads to the same problem as discussed above.

The work of [9] produces a *fault-tolerant* private stream aggregation (PSA) (which is a special case of inner-product MCFE) with a trusted Dealer and the sum functionality. We notice that the idea to achieve fault-tolerance is similar to the dynamic DMCFE. Indeed, the idea of [9] is to run a PSA protocol over all several subgroups in order to get the final result. If a subset of the users fail, they propose to find a disjoint subgroups to cover all the functioning users. The main

[11]Notice that the general definition seems however gives more possibilities and eventually could match with that requirement. To confirm...

challenge is to achieve it with a small number of subgroups. Notice that in the description of their algorithm, since the aggregator needs to find a set of subgroups *covering* the functioning users, it could eventually learn a particular input (a subgroup of size 1). However, we remark that they use differential-private mechanism to ensure that this information is not significant to the aggregator.

We will use all the discussed ideas to produce the PAPAYA functional encryption.

Solving item 2. All previous constructions do not consider the second point. However, one idea is to use a general two-party computation between $\mathcal{E}$ and $\mathcal{R}$ for the decryption procedure. Note also that it is not clear how this situation prevents the initial concern about the privacy of users if there is a collusion between $\mathcal{E}$ and $\mathcal{R}$. For example, using the homomorphic-based solution, the $\mathcal{E}$ and $\mathcal{R}$ could use a two-party decryption protocol over one ciphertext, thus breaking the privacy of the user's input.

5.4 Description of our Solution.

In our context, we do not need the full power of (variants) MCFE but will consider the following idea/constraints in order to build the PAPAYA functional encryption.

- Each user sends one message in a complete decentralized fashion without knowing in advance which other users are going to participate.
- The executor $\mathcal{E}$ acts only as an aggregator and should not learn any additional information.
- The requestor $\mathcal{R}$ is the only entity that gets the final result.
- The scheme needs to be fault-tolerant, i.e if one user fails, the final result should only be considered over the remaining users and no partial information should be given.
- Eventually, we could consider collusion between $\mathcal{E}$ and $\mathcal{R}$.

To resume, the main challenge of our solution is to build a protocol implementing a *decentralized sum protocol robust to fault-tolerance*. In particular, our solution combines the construction of dynamic DMCFE for different subgroups and the fault-tolerant idea of the PSA scheme. Remark that we only need a particular case of the inner-product functionality.

Summary. We review the basic ideas. Each user takes its input x_i and generate a secret key s_i. Denote by $\mathsf{SumDMCFE}$ a restricted MCFE for the sum function, i.e an inner-product DMCFE where only keys for the $1's$ function is allowed. Notice that it can be seen as a *decentralized version* of PSA without a trusted Setup.

The starting point is the *binary tree* idea of Chan et al. [9] that we will review. First, we split users in different groups and run a $\mathsf{SumDMCFE}$ protocol for each group. If some users fail, we can find a set of subgroups that will *cover* the functioning users. This will give the fault-tolerant solution.

5.4.1 Defining the groups.

Suppose that there is at most n users, where $n = 2^m$ for an integer m. We will use the same notations as the PSA protocol. For any integers $k \geq 0$, $l \geq 1$, consider the following sets of indexes B_j^k, called *blocks of rank* k in the original paper, where $B_j^k := \{2^k(j-1)+l, 1 \leq l \leq 2^k\}$. In particular, for n users, denote by $\mathcal{T}(n)$ the blocks B_j^k included in the interval $[1,n]$ for all $k \geq 0, l \geq 1$, i.e

$$\mathcal{T}(n) := \{B_j^k \subseteq [1,n], k \geq 0, l \geq 1\}.$$

Remark that the cardinal of this set is at most $2n$. Next notice that:

- each $i \in [1,n]$ is present in at most one block of rank k; and
- if $[s,t]$ represents a contiguous interval of integers within $[1,n]$, it is computationally easy to find set of at most $2\lceil \log_2 n \rceil + 1$ blocks that uniquely cover $[s,t]$, i.e a set of blocks $\mathcal{B} \subseteq \mathcal{T}(n)$ where each element $i \in [s,t]$ appears in exactly one block in $\mathcal{B}$.

5.4.2 Generalizing fault-tolerant PSA

Forget for the moment the requestor $\mathcal{R}$. We can deduce a general algorithm for our fault-tolerant sum as follows. We fix the set $\mathcal{T}(n)$ and an identifier to each user with an element i in $[1,n]$.

- Each user considers the set $B(i) \in \mathcal{T}(n)$ of blocks where it appears. Then, it could generate a secret key $sk_{i,B}$ corresponding to each $B \in B(i)$ with the Setup algorithm of the SumDMCFE scheme and the consideration of other users of the same B.
- Using the encryption algorithm of the SumDMCFE scheme on inputs $(x_i, sk_{i,B})$, it generates ciphertexts for its input x_i corresponding to each block B, i.e elements $c_i := \{c_{i,B} := \mathsf{SumDMCFE.Enc}(x_i, sk_{i,B})\}$
- In addition, it runs the key generation algorithm of SumDMCFE to produce a part of the functional key for the sum function over each block, i.e

$$dk_i := \{dk_{i,B} := \mathsf{SumDMCFE.UKeyGen}(sk_{i,B})\}$$

- The executor $\mathcal{E}$ receives the contribution of each participating user. Then, it consider a set $\mathcal{B}$ of blocks $B \in \mathcal{T}(n)$ that cover them. Using the above functional keys and the ciphertexts, it could obtain partial sums for each individual block, i.e $\sum_{i \in B} x_i$ for $B \in \mathcal{B}$. The final result is obtained by summing together the different resulted partial sums.

One caveat of this solution is in the last step, where $\mathcal{E}$ could learn each partial sum. In particular, by construction it could happen that some blocks will only represent one user, thus it is easy to recover its input. In the PSA protocol, the use of differential privacy prevents from learning this particular value.

In addition, recall that in our setting, we want to ensure that the requestor $\mathcal{R}$ gets the final result. However, anyone that could learn parts of the functional keys learn (the partial) resulting sum.

Considering $\mathcal{R}$. One solution to overcome this problem is to encrypt the functional parts dk_i using the public key of $\mathcal{R}$. In particular, using its associated secret key $sk_{\mathcal{R}}$, it could recover all the interesting parts (corresponding to blocks of functioning users), then recombine the functional key in order to get the final sum. However, we do not know how $\mathcal{E}$ should proceed in that case. One interesting path is to consider the possible homomorphism in the encryption/keygen algorithms $\mathsf{SumDMCFE.Enc}$ or $\mathsf{SumDMCFE.UKeyGen}$. As in the fault-tolerant PSA protocol, the executor $\mathcal{E}$ could exploit this homomorphism in order to get a ciphertext corresponding to the final sum over the functioning users. Then, the requestor $\mathcal{R}$ obtains by only decrypting the final result.

About the Dsum Functionality for groups. Note that in Dynamic DMCFE, the proposed DSum functionality gives the capability to restrict the sum computation on a chosen group of users. Note however that it does not tolerate faulty users. One solution is to leverage the binary tree idea and incorporate it in a natural way with this Dsum functionality.

6 Conclusion

In this report, we have introduced the specifications of our primitives for privacy preserving data analytics. First, we have presented our solutions for the privacy preserving inference of neural networks. Additionally, we have run experiments for the proposed solutions using different neural network models. Our experiment result showed that 2PC-based solution provides better results while classifying a single input. However, this solution suffer from the burden of the communication cost. On the other hand, FHE-based solution provides promising results in terms of the communication cost and when the input is processed in batches. Moreover, our hybrid solution combines 2PC and FHE to use the advantages of both schemes. Although our experiment results does not show significant improvements for the hybrid solution, it could be easily seen that the increase rate on the computational cost is lower than the other solutions without supporting packing.

For the privacy preserving training of neural networks, we have introduced a solution based on FHE. In this work, we have presented a proof of concept implementation particularly designed for the training of a neural network of a text classifier. Our experiment results show that although the impact on the loss of accuracy is minimal, the impact on the performance is large when compared to train a plain network. Moreover, the training can still be realized on a powerful computer in a reasonable time. Therefore, our future work is to improve the performance of this solution to handle performance drawbacks.

For the privacy preserving collaborative training, we investigate the use of differential privacy and adversary loss to protect against well-known attacks. We have introduced that property inference attack may work on such a collaborative training scenario. We have also presented the preliminary version of our countermeasures specifically proposed for this attack.

For the privacy pre trajectory clustering, we have designed two solutions. The first one is based on the original implementation of TRACLUS algorithm by approximating distance metrics and the second one is based on the use of MinHash algorithm together. Both solutions use 2PC as the cryptographic primitive. Our experimental results show that both solutions are able to cluster of hundreds of trajectory in a reasonable time. As a future work, we are planning to improve the performance of these solutions.

For the privacy preserving counting, we designed a solution based on Bloom filters that employs homomorphic encryption and 2PC as cryptographic primitives. Our experimental results show that the solution can count one million elements less than a minute. Our next goal is to use this solution for the validation of Use Case 3 by integrating it into an real life application.

Finally, we have designed a solution for the privacy preserving basic statistics. This solution uses functional encryption as a cryptographic primitive. As a future work, we plan to complete the implementation of this particular solution.

The validation of use cases using the PAPAYA primitives presented in this document will be given in the upcoming deliverables D5.1 and D5.2.

References

[1] Crypten is a framework for privacy preserving machine learning. `https://github.com/facebookresearch/CrypTen/`.

[2] HElib An Implementation of homomorphic encryption. `https://github.com/shaih/HElib`.

[3] Simple encrypted arithmetic library (SEAL). `https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/`.

[4] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS 2016*, 2016.

[5] Tiziano Bianchi, Alessandro Piva, and Mauro Barni. Composite signal representation for fast and storage-efficient processing of encrypted signals. *IEEE Trans. Information Forensics and Security*, 2010.

[6] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 103–112. ACM, 1988.

[7] Tom Bohman, Colin Cooper, and Alan M. Frieze. Min-wise independent linear permutations. *Electr. J. Comb.*, 7, 2000.

[8] Necati Cihan Camgöz, Ahmet Alp Kındıroğlu, and Lale Akarun. Sign language recognition for assisting the deaf in hospitals. In *Human Behavior Understanding*, 2016.

[9] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 200–214, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Heidelberg, Germany.

[10] Li-Fen Chen, Hong-Yuan Mark Liao, Ming-Tat Ko, Ja-Chen Lin, and Gwo-Jong Yu. A new LDA-based face recognition system which can solve the small sample size problem. *Pattern Recognition*, 2000.

[11] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with repetition for inner product. *IACR Cryptology ePrint Archive*, 2018:1021, 2018.

[12] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[13] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, 2001.

[14] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, pages 1–15, n.a., 2015. The Internet Society.

[15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.

[16] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[17] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML 2016*, volume 48, pages 201–210. JMLR.org, 2016.

[18] Shai Halevi and Victor Shoup. Algorithms in helib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014.

[19] Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine*, 25(1):65, 2019.

[20] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 1973.

[21] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[22] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. *arXiv preprint arXiv:1801.05507*, 2018.

[23] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 1983.

[25] MSc. S. (TNO & Radboud University); Sappelli MSc M. (TNO & Radboud University) Kraaij, Prof.dr.ir. W. (Radboud University & TNO); Koldijk. The swell knowledge work dataset for stress and user modeling research. dans., 2014.

[26] Hans-Peter Kriegel and Martin Pfeifle. Density-based clustering of uncertain data. In *SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2005.

[27] Ken Lang. Cmu text learning group data archives - 20_newsgroup, 2000. `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html`.

[28] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: A partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 593–604, New York, NY, USA, 2007. ACM.

[29] Chae Hoon Lim. Efficient multi-exponentiation and application to batch verification of digital signatures. *Unpublished manuscript, August*, 2000.

[30] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *ACM CCS 2017*, pages 619–631. ACM, 2017.

[31] M. Mansouri, B. Bozdemir, M. Önen, and O. Ermis. PAC: Privacy-preserving arrhythmia classification with neural networks. In *FPS*, 2019.

[32] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 691–706. IEEE, 2019.

[33] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards deep neural network training on encrypted data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[34] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[35] Peter Rousseeuw. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *J. Comput. Appl. Math.*, 1987.

[36] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[37] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *53rd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2015, Allerton Park & Retreat Center, Monticello, IL, USA, September 29 - October 2, 2015*, pages 909–910. IEEE, 2015.

[38] Suman Srinivasan, Haniph Latchman, John Shea, Tan Wong, and Janice McNair. Airborne traffic surveillance systems: Video surveillance of highway traffic. In *International Workshop on Video Surveillance & Sensor Networks*, 2004.

[39] S. Joshua Swamidass and Pierre Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *J. Chem. Inf. Model.*, 47(3):952–964, 2007.

[40] G. Tillem, B. Bozdemir, and M. Önen. SwaNN: Switching among cryptographic tools for privacy-preserving neural network predictions. Technical report, 2020.

[41] A. Wahab, S.H. Chin, and E.C. Tan. Novel approach to automated fingerprint recognition. *IEE Proceedings - Vision, Image and Signal Processing*, 1998.