Towards real-time hidden speaker recognition by means of fully homomorphic encryption

Martin Zuber, Sergiu Carpov, Renaud Sirdey

CEA, LIST,
91191 Gif-sur-Yvette Cedex, France
email: name.surname@cea.fr

Abstract. Securing Neural Network (NN) computations through the use of Fully Homomorphic Encryption (FHE) is the subject of a growing interest in both communities. Among different possible approaches to that topic, our work focuses on applying FHE to hide the model of a neural network-based system in the case of a plain input. In this paper, using the TFHE homomorphic encryption scheme, we propose an efficient fully homomorphic method for an argmin computation on an arbitrary number of encrypted inputs and an asymptotically faster - though levelled - equivalent scheme. Using these schemes and a unifying framework for LWE-based homomorphic encryption schemes (Chimera), we implement a very time-wise efficient, homomorphic speaker recognition scheme using the neural-based embedding system VGGVox. This work can be generalized to all other similar Euclidean embedding-based recognition systems. While maintaining the best-of-class classification rate of the VGGVox system, we implement a speaker-recognition system that can classify a speech sample as coming from one of a 100 hidden model speakers in less than one second.

1 Introduction

An homomorphic encryption scheme is, ideally, an encryption scheme permeable to any kind of operation on its encrypted data. With any input x and function f, with E the encryption function, we can obtain $E(f(x))$ non interactively from $E(x)$. This scheme is called fully homomorphic and we speak of a Fully Homomorphic Encryption (FHE) scheme if its parameters can be chosen independently of the depth of the homomorphic computation to apply. Otherwise we speak of a Levelled Homomorphic Encryption (LHE) scheme.
From the birth of "privacy homomorphism" in [32] to the present day, and through the first truly FHE scheme Gentry presented in [22] there have been many improvements in the field of homomorphic encryption. However, real-world practical application examples are scarce due essentially to limitations in terms of its time and space costs. One such practical application concerns Artificial Neural Networks (ANNs). These are varied and powerful tools that are now ever more present in a wide variety of fields (medical diagnosis methods, autonomous vehicles, financial decision making..), the list of which would be long enough to fit the size of this article. This enthusiastic emergence has, in part, given way to privacy and confidentiality concerns regarding

the vast size of data-sets used for the training of such objects and regarding the type of data that will be "classified" in the inference phase, and by whom.
In this context, FHE is emerging as a possible general and malleable answer to some of those concerns. Indeed, given an ideal FHE scheme, one could in principle chose to hide any type of data at any point in the process of creation and application of an ANN: the training data during the training phase; the classification data during the inference phase; the network itself because it can be expensive to train. The list goes on but the actual practical applications are so far quite few and limited due both to the previously mentioned FHE limitations and the increasing complexity and depth of the state-of-the-art ANNs. We find that focusing on specific types of ANNs with FHE-friendly structures and making use of the wide variety of tools that the FHE research has produced can yield practical and real-world applications, as it does in this paper.

Prior work. Research on the application of techniques for computing over encrypted data, FHE or others "competing" techniques, to ANN-related issues is only at its beginning and has so far barely scratched the surface of the problem. Indeed, the first attempts at applying homomorphic encryption techniques to ANN have all focused only on the inference phase and more specifically on the problem of evaluating a public (from the point of view of the computer doing the evaluation) network over an encrypted input (hence producing an encrypted input). The first work of this kind is CryptoNets [38] where the authors successfully evaluate an approximation of a simple 6 layers Convolutional NN able to achieve 99% success recognition on the well-known MNIST hand-written digits database. Their implementation uses the FV FHE scheme [21] and achieves network evaluation timings of around 4 minutes on a high-end PC. Yet, thanks to the SIMD/batching property of FV-like schemes, one network evaluation can in fact lead to 4096 evaluations of the network done in parallel on independent inputs (i.e. the network is evaluated once on ciphertexts which have many "slots" and thus contain different cleartexts). So, although the latency remains of 4 minutes, the authors rightfully claim their system to be able to sustain a throughput of around 60000 digit recognitions per hour. In subsequent papers, Chabanne et al. [9], [8] are building approximations with small multiplicative depth of networks with up to 6 nonlinear layers. Through significant hand-tuning of the learning step of their networks, they show that these can achieve state-of-the-art prediction quality on both hand-digit (MNIST) and face recognition. However their work lacks an implementation and, hence, they did not provide FHE evaluation timings. More recently, Bourse et al. [5], have fine-tuned the TFHE cryptosystem towards a slight generalization of BNNs (Binary Neural Networks) called DiNNs in which the nonlinear activation function is the sign function which they intricate with the TFHE bootstrapping procedure for more efficiency. Overall, they are able to evaluate small such networks (100 neurons and only one hidden layer) in around 1.5 seconds resulting in a (just decent) 96% prediction accuracy on the MNIST database. As already emphasized, all the previously mentioned papers focus only on the inference phase. Public work on applying FHE to the training phase and on ANN techniques other than mainstream Convolutional NNs are nonexistent with only some works focusing on basic clustering [12], [27] and some focusing on logistic regression model learning [3], [7], [28]. It should also be mentioned that the applications of other

"competing" techniques for computing over encrypted data, the main of which being Secure Multiparty Computations (MPC), to ANN also start to be investigated in their associated communities (e.g., [2], [33]). Additionally, most previous works tackle the problem of evaluating a public network over an encrypted input. Of course, this is an important first step and a pragmatic angle of attack but, even when one limits oneself to the inference phase, other setups are worth investigating such as, for example, that of evaluating a private network or model over a public (again from the point of view of the computer doing the evaluation) input which is also very relevant in many practical situations, as illustrated in the present paper. Yet, just for that first case, the above state-of-the-art demonstrates that no fully satisfying solution has yet been found but also reveals an emerging research trend towards looking for adapted neural network structures. This is where our work aims to fill the gap: by providing a fresh look at how using a specific yet state-of-the-art type of neural-based speaker recognition system can allow us to implement a very time efficient public classification over a private and secure model. Producing a solution to this specific, embedding-based system (see section 2 for details) requires us to solve the nearest neighbour problem (given a number of encrypted inputs, find the one closest to the new, plain, input) in a fully homomorphic way. Furthermore, we also provide a solution to the more general k Nearest Neighbours (k-NN) problem (this time, a solution limited in the number of inputs). Previous work on a subject that can be mapped to secure nearest network computations are [19], [34], [26], [20] and [18]. All of these works (except for [18]) use an additive homomorphic encryption scheme for a distance computation. Then the comparison is done through various means: bit-based approaches that are quite heavy computationally; garbled circuits (see [39]) or other kinds of exchange protocols between the server running the computations and the owner of the data. This is true also for the works solving the general k-NN problem securely. [10] is one such work that uses homomorphic encryption for the distance computation and garbled circuits for the comparison. We can cite [37] as a solution that uses exclusively FHE (the HElib library [25] implementing the BGV FHE scheme [6]) - as we do - to solve the k-NN problem and achieves its results through a bit-wise approach that is significantly more expensive time-wise than our own but, very importantly, for the general k-NN problem, they are not constrained by the number of inputs.

Our contribution. In this paper we consider a state-of-the-art, best-in-class, speaker-recognition embedding system based on convolutional neural networks: VGGVox [17]; the properties of which allow us to implement a very time-wise efficient FHE classification. For this we use the results from [4] which develops a unified, consistent theoretical framework for several homomorphic schemes. It allows us to design both a fully homomorphic and a levelled scheme for speaker recognition based on the neural embedding system from [17]. We implement this with the TFHE library ([15]) based on [13], [14] and the SEAL library [36] using the BFV scheme from the original paper [21]. In the end, our fully homomorphic scheme can classify an audio sample as coming from one of 100 hidden speakers in less than a second.
In the process of building this homomorphic speaker-recognition scheme, we present a general and efficient, fully homomorphic, argmin computation scheme from an arbitrary number of encrypted inputs. In parallel, we present an asymptotically faster,

though levelled, equivalent scheme. For instance, can determine the minimum among 330 encrypted values in less than a second.
Moreover, we build fully homomorphic scheme solving the k-Nearest Neighbours problem for up to 65 encrypted inputs, in less than 0.4 seconds for any number k.

Paper structure. The paper is structured as follows. In section 2, we present the VGGVox neural-based embedding system on which we base our fully homomorphic classification scheme and give a case study of a possible application of our work. Section 3 presents the homomorphic schemes and operations on which we base our work. Section 4 presents the general algorithms we designed to provide homomorphic classification schemes while section 5 shows how and to what extent those schemes can be considered practical: we present the parameters used as well as the timings we obtain and the classification precision we achieve.

2 A Neural Embedding system: VGGVox

VGGVox. What we call VGGVox in this paper refers to a neural embedding system presented in [17]. It was trained and tested on datasets called VoxCeleb1 [30] and VoxCeleb2 [16]. We refer to the cited papers for an in-depth presentation of the system. We present here enough material from these works so that it is clear exactly what and where is our contribution.
The system applies two consecutive Convolutional Neural Networks (CNN) to a raw audio segment containing a speech sample from one given speaker. The second of these networks outputs a real vector embedding of dimension 1024 that is a representation of the speaker in the Euclidean space $\mathbb{R}^{1024}$. The point of the system is to make it so the Euclidean distance in the output space ($\mathbb{R}^{1024}$) becomes a good measure of the similarity between the speakers of the samples in the input space (raw audio samples). In short, if we find that two audio samples going through the networks yield two vectors that are "close", then we can assume that the same speaker has provided the two original samples. The way that we use this system is to have a certain number of "model" embeddings each representing a different speaker. We classify the new sample as being spoken by one of the model speakers by finding the model embedding that is closest to the new embedding. Therefore, we need to solve a nearest neighbour problem.
This kind of embedding-based neural architecture is not universally applicable. However, it has shown to be a very efficient way to solve specific problems. One such system is used in [35] for face recognition but embedding-based systems are also typical for text-processing applications as is stated by Goodfellow et al. in [24]. Also, [35] and [17] show that the field of embedding-based neural systems is evolving quickly and we can expect that other applications will be found in the years to come. This is important because the work that we provide in this article can be generalized to any system working in the same way as the VGGVox system. As long as the measure of similarity that interests us at the output of the neural networks is the Euclidean distance, then the framework we present works without having to be adapted to the specifics of the new system.
We refer to [17] for precise information on the performance of VGGVox (in terms

of classification rate), but we feel confident in stating that it is a highly accurate speaker-recognition system that achieves best-of-class performance.

Case study. There are several applications that can be thought of to apply the results from the VGGVox system. In our case, we set out to use its embedding feature to our advantage. The main idea is to provide a degree of protection for the identity of the model speakers used in a speaker-recognition application, without having to apply homomorphic encryption to very deep CNNs, and in a practical, time-efficient manner. The following (illustrated in figure 1) is an example of application to show how and why our scheme could be implemented in the context of a real-world application.
We place ourselves in the case where, at a given location, George - "the gate" - owns a microphone device that is made to capture audio samples, to determine who is or is not present on site. Bianca - "the boss" - needs to know, through speaker recognition, who is getting in and out of the location. George cannot have access to the model vectors associated to the employees, because the gate is not a secure location for data storage. Bianca must only have the information that she is due: the identity of an employee going through the gate, but not the recording of their voice.
The audio samples taken by George are run through the VGGVox system in plaintext, on location. The embedding that it yields is then compared homomorphically to a number d of stored, homomorphically encrypted, "model" embeddings that are provided by Bianca [1] in order to determine which of the stored embeddings is closest to the newest one. The result of this homomorphic comparison is sent to Bianca. Bianca, owns the key to the private homomorphic encryption scheme and can decipher the result sent by George.
At this point Bianca knows which person has used the microphone at the secure location but does not have any access, at any point, to the recordings themselves. At the same time George does not have access to the model recordings which are homomorphically encrypted with a key he does not own.

3 Homomorphic Operations and Noise propagation

3.1 LWE encryption schemes

Notation. We use $\mathbb{B}$ to denote the set $\{0,1\}$. We denote signed integers by $\mathbb{Z}$, the reals by $\mathbb{R}$ and use $\mathbb{T}$ to denote the real torus mod 1. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ will denote the integers mod q with a given integer q. The ring $\mathbb{Z}[X]/(X^N+1)$, where N is a fixed integer, is denoted $\mathfrak{R}$ and the ring $\mathbb{Z}_q[X]/(X^N+1)$, $\mathfrak{R}_q$. $\mathbb{B}_N[X]$ is the subset of $\mathfrak{R}$ composed of the polynomials with binary coefficients. We write $\mathbb{T}_N[X]$ the quotient $\mathbb{R}[X]/(X^N+1)$ mod1. Vectors are denoted with an arrow as such: $*$. Ciphertexts are denoted with either boldface letters or brackets ($[*]$). We write $||\cdot||_p$ to denote the ℓ_p norm of vectors over reals or integers. In the following, λ is the security parameter.

[1] George could also get the model embeddings on his own, at the first passage of a new hire, and then encrypt the new embedding with a public key provided by Bianca. This way Bianca would not have access to the model embeddings either.

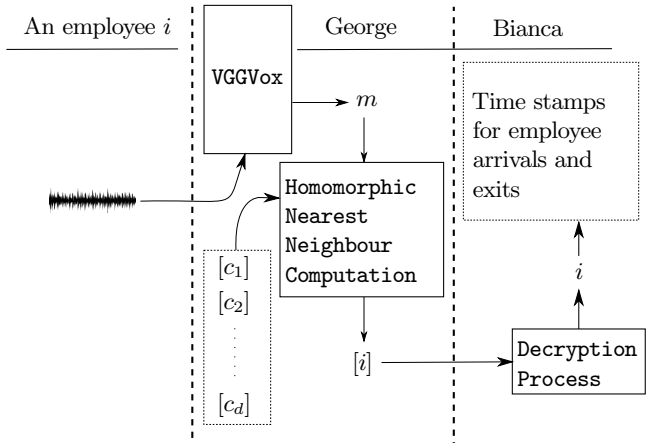


Fig. 1. A figure illustrating the case study that we provide. Here the d model embeddings are represented by the variables c_1 to c_d and the encryption function is represented by brackets $[*]$. When an employee goes through the gate, he provides a speech sample. The sample is embedded into the vector m by the VGGVox system and then homomorphically compared to the stored, encrypted model embeddings. An encryption of the identity of the employee is then sent to Bianca for decryption and storage.

Distribution. If $\mathcal{D}$ is a probability distribution, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled according to $\mathcal{D}$. The Gaussian distribution over $\mathbb{R}$ centered at 0 and of parameter $\sigma \in \mathbb{R}^+$ (denoted $\mathcal{N}(0,\sigma)$) has a density function proportional to $\rho_\sigma(x) = \exp\left(-\pi||x||^2/\sigma^2\right)$. In the rest of the paper, when not specified otherwise, we use a Gaussian distribution when sampling randomly. For instance, the distribution χ_e will be the Gaussian distribution we sample an error e from.

LWE problem. We recall the LWE problem introduced by Regev in [31]. We let n be an integer that sets the dimension of the problem and we let χ_e to be a noise probability distribution over $\mathbb{R}$. The LWE assumption states that for a secret $\boldsymbol{s} \in \{0,1\}^n$, it is computationally hard to distinguish between the two following distributions:

- the LWE distribution which outputs $(\boldsymbol{a},b)$ where $\boldsymbol{a}$ is taken uniformly at random from $\mathbb{Z}_q^n$ and $b = \boldsymbol{s}\cdot\boldsymbol{a} + e$ with $e \leftarrow \chi_e$.
- the uniform distribution over $\mathbb{Z}_q^{n+1}$.

There is a "ring" version, ring-LWE, which was introduced in [29].

LWE based encryption schemes. We mention the LWE problem here because both of the encryption schemes that we use are based on it. They are the BFV ([21]) scheme, implemented in the SEAL ([11], [36]) library and based on the ring-LWE problem, and the TFHE scheme ([13], [14]) implemented in the TFHE library ([15]) and based

on both the LWE and ring-LWE problems. We will not go into the details of any one of these schemes and refer the reader to the original papers. We will only present here the information necessary to understand the schemes that we build as they are presented in section 4.
In essence, since the work in [4], the two encryption schemes can be considered to be the same one (both part of a bigger, all-encompassing scheme: Chimera). Ciphertexts in BFV are elements of $(\mathfrak{R}_q)^{k+1}$ and they encrypt values taken from $\mathfrak{R}_p$ with q the ciphertext modulus, p the plaintext modulus and k a given integer parameter. In their ring-LWE variant, ciphertexts in TFHE are in $\mathbb{T}_N[X]^{k+1}$ and encrypt values in $\mathbb{T}_N[X]$. TFHE can also encrypt scalar values in $\mathbb{T}$ with ciphertexts in $\mathbb{T}^{n+1}$ where n is an integer parameter (the scalar equivalent of $k \times N$). In both cases the secret key can be considered to be taken from $\mathbb{B}_N[X]^k$ (respectively $\mathbb{B}^n$ in the scalar case). In the ring-LWE case, we can see that up to a rescaling factor (by q for ciphertexts and p for plaintexts) these two schemes can be considered to have the same plaintext and ciphertext spaces. [4] shows that the ciphertexts from one scheme can in fact be homomorphically transformed into ciphertexts from the other scheme. Since we do not use packing in our work here [2] , we can go from one scheme to the other instantly and noiselessly.
From now on, we are therefore not going to identify a ciphertext by its associated encryption scheme but rather by whether it encrypts a polynomial or a scalar value. Therefore a ciphertext of a polynomial $m[X]$ will be written as $[m[X]]^{(r)}$ and one for a scalar m as $[m]$. Similarly, we assume here that we can apply any operation on a given ciphertext that can be applied in SEAL and TFHE. The plaintext space and ciphertext space used will be considered to be the torus ones for simplicity sake, unless specified otherwise.
One exception to this rule is the `TRGSW` encryption. `TRGSW` is the ring version of the GSW scheme introduced in [23]. A `TRGSW` ciphertext encrypts messages in $\mathfrak{R}$. We only ever use this encryption scheme as way to apply a specific MUX gate. For that purpose, the message space of `TRGSW` ciphertexts in this paper will only ever be $\{0,1\}$ (polynomials of degree 0). This means we will represent `TRGSW` ciphertexts as scalar ciphertexts $[*]$. We refer to section 3.2 for more details.

3.2 Homomorphic Operations

In this section we will present the different operations in TFHE and SEAL and their noise propagations. Since they are based on the LWE (or ring-LWE) problem, both the SEAL and TFHE encryption schemes rely on a noise to be introduced in the ciphertext. We assume that it is a Gaussian noise unless stated otherwise and refer to it through its standard deviation that we write as σ in most cases. Again, the noise is the same in TFHE and SEAL up to a rescaling by q. Importantly, whenever we present a value for a standard deviation in the paper, it will be one rescaled and therefore applicable in the torus. To use it in a SEAL ciphertext, multiply by q. With every homomorphic operation (except the bootstrap operation as presented in section 3.2) the noise grows. This is

[2] [4] uses a time-expensive public key-switch operation to switch from packed SEAL ciphertexts to unpacked TFHE ciphertexts. We do not use it in this work.

a fundamental issue in FHE research in general and in this paper in particular. We will refer to it as "noise propagation". We add to the notation of the ciphertext a possible mention of an encryption key s and a noise α as such: $[*]_{s,\alpha}$. These operations are all operations that are implemented in either the SEAL library or the TFHE library. In fact, all of the operations have a TFHE version while only the first three are implemented using SEAL. We refer to the corresponding papers for more information.

- Scalar Internal Addition : $[*]\times[*]\to[*]$
  Given two ciphertexts $[\mu_1]$ and $[\mu_2]$, we can add them and obtain a ciphertext $[\mu_1+\mu_2]$. We call this operation $\mathsf{InternAdd}$. The exact same operation exists for polynomial ciphertexts with, notably, the same noise propagation. Therefore we will use the notation $\mathsf{InternAdd}$ for both cases. The operation $\mathsf{InternSub}$ is its subtraction counterpart.
- Scalar External Multiplication : $*\times[*]\to[*]$
  Given a ciphertext $[\mu]$ (scalar or polynomial) and a scalar α, we can multiply them and obtain a ciphertext $[\alpha\times\mu]$. We call this operation $\mathsf{ScalarExternMult}$. As with $\mathsf{InternAdd}$, we will use the notation $\mathsf{ScalarExternMult}$ for both scalar and polynomial ciphertexts.
- Polynomial External Multiplication : $*^{(r)}\times[*]^{(r)}\to[*]^{(r)}$
  Given a polynomial ciphertext $[\mu[X]]^{(r)}$ and a polynomial $\alpha[X]$, we can multiply them and obtain a ciphertext $[\alpha[X]\cdot\mu[X]]^{(r)}$. We call this operation $\mathsf{PolyExternMult}$.
- Extraction : $[*]^{(r)}\to[*]$
  From a ring ciphertext of a polynomial $\mu[X]=\sum_{i=0}^{N-1}\mu_i$, it is possible to extract a scalar ciphertext of a single coefficient of μ_p at a position $p\in[0,N-1]$. We can do this at no cost to the noise of the ciphertext. We call this operation $\mathsf{SampleExtract}_p$. We can extract similarly the corresponding scalar secret key from the initial ring secret key.
- Public Rotation : $[*]^{(r)}\to[*]^{(r)}$
  Given a ring ciphertext of a polynomial $\mu[X]$ and an integer p we can obtain a ciphertext of $\mu\times X^p$ with no noise increase. We call this operation Rotate_p.
- Key-Switch : $[*]_s\to[*]_{s'}$
  From two keys s and s', we can create an object $\mathsf{KS}_{s\to s'}$ (KS sometimes for short) called the key-switching key with a precision depending on parameters base and t. Given a ciphertext $[\mu]_{s,\alpha}$ of a scalar value μ encrypted using the key s with noise α, and this key-switching key KS, we can obtain a ciphertext $[\mu]_{s',\alpha'}$ which encrypts the same value under the key s' and with a higher noise $\alpha'>\alpha$. We write this operation $\mathsf{KeySwitch}_{s\to s'}$.
- Bootstrapping [3] : $[*]_{s,\alpha}\to[*]_{s',\alpha_b}$
  From two keys s and s', we can create an object $\mathsf{BK}_{s\to s'}$ (BK for short) called the bootstrapping key, with a precision depending on parameters ℓ and B_g.

[3] This bootstrapping is only a slight variation on the bootstrapping procedure introduced in [13], we just add a public rotation to the bootstrap operation used in [5].

From these, we define two parameters: $\beta = B_g/2$ and $\epsilon = \frac{1}{2B_g^\ell}$. Given an integer b, a ciphertext $[\mu]_{s,\alpha}$ of a scalar value μ encrypted using the key s with noise α, and this bootstrapping key BK, we can obtain a ciphertext $[\mu_0]_{s',\alpha_b}$ where $\mu_0 = 1/b$ if $\mu \in [0, \frac{1}{2}]$ and $\mu_0 = 0$ if $\mu \in [\frac{1}{2}, 1]$ [4] . Very importantly, α_b is fixed by the parameters of the bootstrapping key BK and does not depend on the initial standard deviation. We write this operation $\mathsf{BootStrap}_{s \to s',\alpha_b}$. We call it "sign bootstrap" because the function that it applies (it could apply other functions) can be considered a sign computation.
This operation therefore allows us to both apply a sign function to the input ciphertext and reduce its noise down to α_b. Figure 2 is a representation of this operation. The application of the function is not infinitely precise. The figure shows how there are values for which the operation does not necessarily output the correct value. This is not a problem when the parameters are chosen accordingly: see section 5.2 for details.

- Circuit Bootstrapping : $[*] \to [*]$ Given a scalar ciphertext $[\mu]$, we can output a `TRGSW` ciphertext $[\mu_0]$ where $\mu_0 = 1$ if $\mu \in [0, \frac{1}{2}]$ and $\mu_0 = 0$ if $\mu \in [\frac{1}{2}, 1]$. This operation is composed of several bootstrap and key-switch operations and outputs the result of a sign bootstrap as seen above, only under a `TRGSW` encryption. We exclusively use this operation in order to then apply a TFHE MUX gate. We call this operation $\mathsf{CircuitBoot}$.
- TFHE MUX gate: $[*] \times [*]^{(r)} \times [*]^{(r)} \to [*]^{(r)}$
Given a `TRGSW` ciphertext $[b]$ of message $b \in \{0,1\}$. Given two ciphertext polynomials $[\mu_1[X]]$ and $[\mu_2[X]]$. We can apply a MUX gate that outputs $[b?\mu_1 : \mu_2]$, which is a ciphertext of μ_1 if $b = 1$ and a ciphertext of μ_2 if $b = 0$. We call this operation MUX.

We present in table 1 the noise overhead of the operations that we use in this work.

4 Homomorphic Speaker Recognition

In order to determine homomorphically which encrypted vector from a number of model embeddings is closest to a plain vector input, first, we need to be able to compute distances, and second, we need to be able to compare those distances. We present in this section the main algorithms for the distance computation and the comparison phase. The initial distance is computed using SEAL ciphertexts and SEAL operations. The distance ciphertexts then become TFHE ciphertext (again, with a simple rescaling and therefore at no significant cost) for the comparison phase. We will first present the distance computation operations and then two schemes for the comparison of all the distances, each with their strengths and weaknesses.

[4] We implicitly write the possible values of μ and the output value μ_0 as members of the torus space $\mathbb{T}$. Alternatively, we also refer to $1/b$ as the value 1. This is arbitrary but allows us to represent the bootstrap operation very intuitively in figure 2.

Operation	Noise
$\mathsf{InternAdd}(c_1,c_2)$	$\vartheta_1+\vartheta_2$
$\mathsf{ScalarExternMult}(\mu,c)$	$\lvert\mu\rvert\cdot\vartheta_c$
$\mathsf{PolyExternMult}(\mu,c)$	$N\lvert\lvert\mu\rvert\rvert(\lvert\lvert e\rvert\rvert+r_p(q)/2)$
$\mathsf{SampleExtract}_p(c)$	ϑ_c
$\mathsf{Rotate}_p(c)$	ϑ_c
$\mathsf{KeySwitch}(c)$	$\vartheta_c+nt\vartheta_{\mathsf{KS}}+n\mathsf{base}^{-2(t+1)}$
$\mathsf{BootStrap}(c)$	ϑ_{boot}
$\mathsf{CircuitBoot}(c)$	$\vartheta_{\text{boot}}+nt\vartheta_{\mathsf{KS}}+n\mathsf{base}^{-2(t+1)}$
$\mathsf{MUX}(C,c_1,c_2)$	$\max(\vartheta_1,\vartheta_2)+A\vartheta_C+B$

Table 1. Elementary operations and their noise overhead Here the c represent ciphertexts while the μ represent cleartexts. C is a `TRGSW` ciphertext. ϑ_i represents the variance of ciphertext c_1 before the operation. The same goes for ϑ_c, the variance of ciphertext c. In the case of $\mathsf{PolyExternMult}$, the noise e of the input ciphertext is a polynomial called "inherent noise" in [11] and $r_p(q)=(q \mod p)$. We refer to that paper for more information. ϑ_{KS} is the variance for the encryption of the key-switching key KS and ϑ_{BK} for the bootstrapping key BK. In the case of the bootstrapping, the output variance overhead is given by: $\vartheta_{\text{boot}}=2Nn(k+1)\ell\beta^2\times\vartheta_{\mathsf{BK}}+n(1+kN)\epsilon^2$. A and B are constants with values $A=(k+1)\ell N\beta^2$ and $B=(kN+1)\epsilon^2$.

4.1 Distance Computation

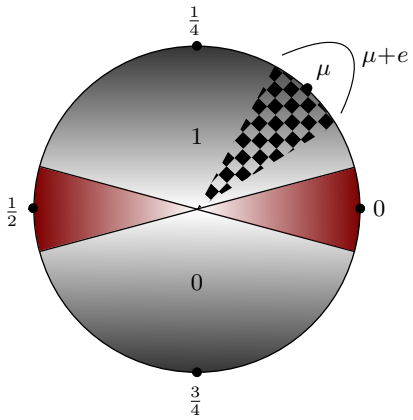
We are given d real vectors: $c^{(k)}\in\mathbb{R}^\gamma, k\in[1,d]$ of dimension γ. These are the outputs of the VGGVox system for d different speakers. To be more precise, we are going to use an approximation of the actual VGGVox output vectors. We are going to have $c^{(k)}\in\mathbb{N}^\gamma$ and we refer to section 5.1 for further information. We encode each of these vectors in polynomial form:

$$\forall k\in[1,d],\quad C^{(k)}=\sum_{i=0}^{\gamma-1}c_{i+1}^{(k)}\cdot X^i$$

Actually, the entity performing the distance computation, as mentioned in section 2, only has an encrypted version of those vectors: $\left[C^{(k)}\right]^{(r)}$. These ciphertexts are the model ciphertexts and they are stored in "the gate" waiting for a new vector to come in for comparison. Let's call $m\in\mathbb{R}^\gamma$ this new vector and encode it as a polynomial:

$$M=\sum_{i=0}^{\gamma-1}m_{\gamma-i}\cdot X^i$$

Given this M and the $\left[C^{(k)}\right]^{(r)}$, we want to compute $\mathbb{d}_k$ the d distances from m to every one of the $c^{(k)}$. This is a basic computation and we only go into detail so that the depth of the computation is clear given the pre-computed values. We define:



*

Fig. 2. This figure illustrates the principle behind the sign bootstrapping function. The circle represents the torus. $\mu \in \mathbb{T}$ is an example of a torus message. When encrypted it gains a Gaussian error e, the checkered slice and curve show the range that it can take. It corresponds to the values that $\mu+e$ will take with probability $1-2^{-64}$. If that range is completely encapsulated within the top part of the circle, then the sign bootstrap function will output an encryption of 1 with a fixed error. The bottom part corresponds to an output of 0. The red parts correspond to the uncertain zones where a lack of precision from the bootstrap operation means the output is uncertain.

$$\mu = \sum_{i=1}^{\gamma} m_i^2 \quad \text{and} \quad \forall k \in [1,d], \quad \mu^{(k)} = \sum_{i=1}^{\gamma} \left(c_i^{(k)}\right)^2$$

For a given k, let's call $\mathbb{D}_k$:

$$[\mathbb{D}_k]^{(r)} = -2 \times M \times [C^{(k)}]^{(r)} + \mu \cdot X^{\gamma-1} + [\mu^{(k)} \cdot X^{\gamma-1}]^{(r)} \tag{1}$$

We can have $-2 \times [C^{(k)}]^{(r)}$ and $[\mu^{(k)} \cdot X^{p-1}]^{(r)}$ be pre-computed. Then

$$[\mathbb{d}_k] = \mathsf{SampleExtract}_{\gamma-1}([\mathbb{D}_k]^{(r)})$$

yields an encryption of the distance between m and the model vector $c^{(k)}$.
It is straightforward to see that this yields the correct result. What is important here is that, given the pre-computations, the distance only requires one external multiplication, one external addition and one internal addition.

Alternative method: As we will see in section 4.3, if we choose to solve the argmin problem using the "matrix" version, we do not actually need to compute the distances themselves but rather all the distance differences $(\mathbb{d}_k - \mathbb{d}_l)$. This means we can do:

$$[\mathbb{D}_k - \mathbb{D}_l]^{(r)} = M \cdot \left[2\left(C^{(l)} - C^{(k)}\right)\right]^{(r)} + \left[\left(\mu^{(k)} - \mu^{(l)}\right) \cdot X^{\gamma-1}\right]^{(r)} \quad (2)$$

with

$$\left[2 \times \left(C^{(l)} - C^{(k)}\right)\right]^{(r)} \quad \text{and} \quad \left[\left(\mu^{(k)} - \mu^{(l)}\right) \cdot X^{\gamma-1}\right]^{(r)}$$

pre-computed for every k and l. Then, as before we extract the $(\gamma-1)$th coefficient, to obtain $[\mathbb{d}_k - \mathbb{d}_l]$ The main drawback here is the amount of data to be pre-computed and therefore stored. However, there are two advantages to doing this. First, we compute one less external addition for every computation (although as is shown in section 5.3 that is not significant time-wise). Secoond, and more importantly, it reduces the range that the underlying encrypted values have, therefore reducing the strain on the parameters (the FV operations can use a lower plaintext modulus p and the TFHE operations will be more accurate).

4.2 The Tree method

At this point, we want to be able to find the index of the minimum distance. There are two ways that we can do this. The first one consists of building two parallel trees: one for the min computation and the other for a parallel argmin computation. We will call this version the "tree version".

The min *tree:* The min tree outputs the overall minimum distance. It compares two distances at every level and then compares the "winners" of the previous level in the current level. We start by computing an indicator δ that indicates which of two distances is the smallest one: for instance, at the first level, for the two distances $\mathbb{d}_k$ and $\mathbb{d}_l$:

$$\begin{aligned} \delta_{k,l} &= 1 \quad \text{if} \quad \mathbb{d}_k < \mathbb{d}_l \\ &= 0 \quad \text{otherwise} \end{aligned}$$

The tool used here is the bootstrap introduced in [13]. With a sign function as output function and an addition to re-calibrate the result we obtain a scalar ciphertext of $\delta_{k,l}$ [5] for any given k and l. Of course 1 and 0 are not the actual torus values. In practice, for every application of the sign bootstrap, we can choose any integer b so that the output is either 0 or $\frac{1}{b}$. The chosen base b will be given for every use of the bootstrap operation but we may also - when convenient - only refer to the two outputs as 1 and 0. In this case for instance, we use a base $b=4$. This δ indicator is then lifted to a `TRGSW` ciphertext through the CircuitBoot operation. We use it in the TFHE MUX gate to select the minimum distance for the next level of the tree. This means that

[5] In the case where $\mathbb{d}_k = \mathbb{d}_l$ the sign bootstrap yields a random output. This is actually also the case when $\mathbb{d}_k$ is "close" to $\mathbb{d}_l$: this means the difference is in the red zone around 0 seen in figure 2. See section 5.1 for implications.

this scheme is not fully homomorphic. Indeed every application of the MUX gate adds noise to a ciphertext that is itself reused in that same MUX gate at the next level of the tree. We found no way to reduce this error over time. Therefore the parameters have to be chosen according to the depth of the tree. By repeating these operations we find ourselves, after the last comparison, with the overall minimum distance.

The argmin *tree:* The argmin tree computes the index of the minimum distance. It uses the δ values created by the min tree to select which index can go through to the next level of the tree. Every index from 0 to $d-1$ is encoded as a polynomial representation of its own base 2 decomposition [6] : for an index k it is the unique polynomial $P_k \in \mathbb{B}[X]$ such that $P_k(2)=k$. The TFHE MUX gate is applied on those indexes with the δ values as deciders. Both trees are presented in figure 3.

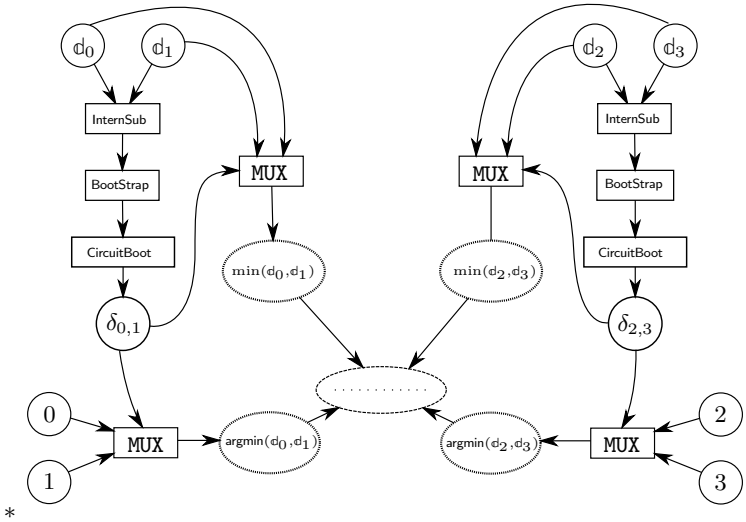


Fig. 3. This figure shows the selection of the min and argmin for the first 4 distances. After the first round of selection, $\min(\mathbb{d}_1,\mathbb{d}_2)$ and $\min(\mathbb{d}_3,\mathbb{d}_4)$ are compared in the second round. This goes on until the final argmin and min values are computed.

Potential alternate tree: As will be shown in section 5.3, this is a somewhat heavy-handed way to solve the problem. Indeed the circuit bootstrapping is the main computational burden and is only used to change the nature of the ciphertext and not to apply any "useful" operation. A way to make this tree much more efficient is to use the SEAL internal multiplication to create a SEAL MUX gate that would only consist in basically two multiplications right after the bootstrapping step. However this is not realistic as of now because of the noise propagation in the SEAL internal multiplication. Given the output bootstrapping noise, the noise grows prohibitively after only one SEAL MUX gate. This could be solved with a higher precision implementation of the TFHE library, allowing for a great noise reduction from the BootStrap operation.

[6] This greatly reduces the stress on the parameters induced from the MUX gate.

4.3 The Matrix method

Another way to go about determining an argmin homomorphically is what we will informally call the "matrix" way. Instead of treating the compared distances two-by-two in a tree, we compute all of the δ values in bulk. This gives us the following matrix:

$$\begin{pmatrix} 0 & \delta_{1,0} & \cdots & \delta_{d-1,0} \\ \delta_{0,1} & 0 & \cdots & \delta_{d-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{0,d-1} & \delta_{1,d-1} & \cdots & 0 \end{pmatrix}$$

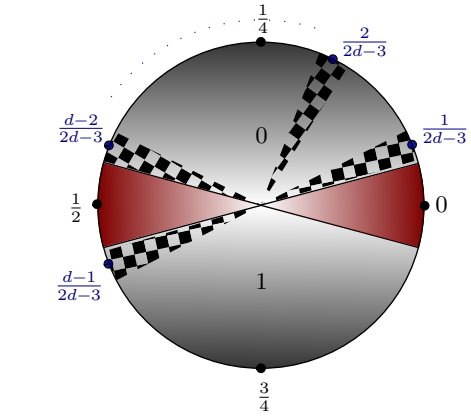
As seen in section 4.2, these are obtained by subtracting the distances to compare and then applying a bootstrap operation. There are effectively $\frac{1}{2}\cdot(d^2-d)$ bootstraps because $\delta_{i,j} = 1-\delta_{j,i}$ for every i,j. At this point, we sum the lines of the matrix together and obtain:

$$\begin{pmatrix} \Delta_0 & \Delta_1 & \cdots & \Delta_{d-1} \end{pmatrix} \quad \text{where} \quad \Delta_i = \sum_{j=0}^{d-1} \delta_{i,j}$$

All of the Δ_i are between 0 and $d-1$. There is only one of them (let's call it $\Delta_{\max}$) with value $d-1$. The index for that Δ is the argmin we are looking for. Indeed, it corresponds to the only distance that is lower than every other one and therefore has δ values always equal to 1. As mentioned previously, these values of 0 and 1 are only theoretical. In practice there is a base b so that the values are actually 0 and $\frac{1}{b}$. Here we set this to $2d-3$. This is important because it is the value which allows us to have $\Delta_{\max}$ be the only value above $\frac{1}{2}$ in the torus circle. As seen in figure 4, this means that applying a sign bootstrap [7] on the Δ values yields a 1 only for $\Delta_{\max}$ and 0 for all other values.

Remark: In practice, depending on the number d of distances to compare, the Δ values cannot be obtained immediately through a single sum and then a bootstrap. There are two limiting factors for the output of the operation to be correct: the noise propagation during the sum, and the precision of the sign bootstrap (as seen in figure 2). Therefore, in practice, we divide the sum in "chunks" of a given number m of ciphertexts and make intermediate sign bootstraps that output 1 if the intermediate sum is maximal and 0 otherwise. Therefore, the base for the first bootstrap operation has to be $\frac{1}{2m-3}$. We give the value we use for m in section 5.2.
This scheme is fully homomorphic because the value m does not depend on the number of inputs to be compared but rather on the parameters chosen initially. Once the parameters are chosen and the value m known, any number of ciphertexts to compare can be divided into chunks of m ciphertexts, summed and bootstrapped, and the operation repeated because after each bootstrap operation, the noise is reinitialized.

[7] This time where $b=4$ and where the 0 and the 1 outputs are swapped



*

Fig. 4. A torus representation of the Δ values. As we can see here, applying a bootstrap on all of them means only one (the highest value) will yield an output of 1. We can see here that compared with figure 2 the 0 and 1 values are swapped. The checkered zones still represent the actual range of the message values. The parameters must be chosen so that they do not overlap with the red zones.

k-NN solution. We can see that, on figure 4, rotating the circle can allow us to have any number of values in its bottom part, and not just one like we do here. Placing every value but two in the bottom part of the circle would actually correspond to selecting the 2 closest vectors to the input vector. The same goes for any number k, giving us a solution to the k Nearest Neighbours problem homomorphically.
However, this can only work if the selection is done through one bootstrap, and therefore if the number of distances is higher than m (the maximum number of values we can add together before a bootstrap) then this k-NN scheme does not work.

5 Practical Implementation

5.1 Precision

The two schemes presented in sections 4.2 and 4.3 do not allow us to perform an exact homomorphic transposition of the VGGVox classifications. This means that the classification rate of our homomorphic system will be lower than the non-encrypted one. This is due to two main factors:

– The precision of the input vectors. The output vectors of the VGGVox networks (the ones we take as inputs for our distance computations) are real numbers. Because of strains on the parameters we cannot afford to match their precision. This means we choose to round the vectors down to a certain number of digits. This strain corresponds to the fact that the values in SEAL (resp. TFHE) must not go above $\frac{p}{2}$ (resp. $\frac{1}{2}$) during the distance computation.

– The precision of the bootstrap operation. As shown in figure 2, the bootstrap operation, given a set of parameters, has a "red" zone around 0 and $\frac{1}{2}$ where input values yield an uncertain output. We can arrange for the values of $\mathbb{d}_k - \mathbb{d}_l$ to never reach the zone around $\frac{1}{2}$ (by reducing the precision of the inputs as seen above). However, if two distances are too close, their difference can be too low for the bootstrap operation to yield the correct result.

Both of these sources of error are manageable. Indeed, the worst outcome is misclassifying a speaker as the wrong one, albeit one close to the good answer. Importantly, it does not matter if the scheme fails at computing the correct sign for distances which do not involve the final minimum distance. It will not change in any way the final result. Our only condition is that the comparisons that involve the distance which is actually the minimum be correct. We tackle this issue by trying to estimate the new classification rate in section 5.3.

5.2 Parameters

We set the parameters according to two constraints: the accuracy of the final result and the security of the overall scheme.

Security: We base the security of our scheme on the `lwe-estimator` [8] script. The estimator is based on the work presented in [1]. It allows us to find the smallest initial noise for our ciphertexts, that still ensures security and gives us the most leeway in terms of noise propagation. The security, in FV, depends not only on the ciphertext noise but on the value of the ciphertext modulus (q) and the degree of the polynomials (N). There is no coefficient modulus in TFHE. Table 2 in appendix A shows minimum noise values for a given set of parameters. We restrict ourselves here to two security parameter values: 80 and 110. However, the precision for the SEAL (resp. TFHE) library is up to 7.6e−9 (resp. 1.11e−16) as of now. By this we mean the lowest value that can be taken as a standard deviation for a Gaussian sampling. Thus, any standard deviation below that could not yield an actual implementation. Therefore, we take the highest σ of the two when a conflict occurs.

Accuracy: As mentioned in section 5.1, whichever way we choose to do it, there is an inherent approximation in the homomorphic classification that we compute. In our case, we choose to use an approximation factor of 3 for the vector inputs: we truncate the values after the third digit. The parameters we use for the initial SEAL distance computation are given in table 3 in appendix A. The value of p needs to be high enough to prevent the distance values from "overflowing" and needs to verify $q = 1 \mod p$ in order to reduce noise propagation. When setting the parameters for the bootstrap and key-switch operations of the min and argmin computations, one needs to strike a good balance between efficiency and accuracy while taking security constraints into account. This leads us to the parameters presented in table 4, in appendix A.
Furthermore, with the given parameters, in the matrix version, we can add up to

[8] https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py

65 δ values together before applying a BootStrap operation (as seen in section 4.3). Any more than that, and we could not certify the result of the output. As we note in section 4.3, this means that our paper proposes a fully homomorphic scheme for solving the k-NN problem for up to 65 values.
We mentioned before that the tree scheme is levelled: this means that the given parameters here only allow us to build a tree with a certain given depth. This is not true for the matrix scheme which can, with the given parameters, be used for a classification among an arbitrary number of embedding models. For the tree scheme, the depth that these parameters allow us to go to, in this case, is 72. This means we can classify among 2^{72} different model embeddings. In practice, given a fixed number of models beforehand, one could reduce the size of the parameters we give here to fit the size of their tree and increase the time performance of their classification.
Overall, with these parameters, we achieve a 127-bit security in the "matrix" scheme and an 80-bit security in the "tree" scheme. In the "tree" scheme, setting parameters to obtain a higher security level would compromise the usefulness of the network in that it would greatly limit the number of comparisons we can make, or make a single comparison operation prohibitively expensive in terms of time and storage.

5.3 Performance

Classification rate: As we stated in section 5.1, our homomorphic distance and argmin computations do not match the necessary precision to replicate exactly the classification results from the VGGVox system in [17]. In this paragraph we will strive to evaluate how that affects the classification rate of our homomorphic equivalent given that we have only built a prototype and not fully implemented it, and therefore cannot run the classification tests directly. Given our set of parameters, we can determine that in order for the BootStrap operation to reliably differentiate between two distances, their difference must not be greater than 2.51e−02. Below such a value, the operation will select one or the other randomly. This allowed us to simulate our homomorphic scheme in the clear by testing with several test embeddings whether, given the shortest distance, the difference between that distance and the others was below 2.51e−02. As we saw in section 5.1, it is the only condition we need for the overall argmin computation to be exact. Experimentally, we find that after truncating test vectors at the third digit (as seen in section 5.2) and running the tests on several groups of embeddings, if the VVGVox system classified a recording correctly, then the smallest distance was smaller than the other distances by a wide margin (at least 0.1). However, a serious evaluation of the impact of the application of our scheme on the classification performance of the system would require systematic testing that we do not provide in this work. We conclude that our precision is good enough that our system will classify correctly most of the times that the plain one does. But we assume that we still incur a loss in classification performance that we cannot evaluate precisely at this point.

Timings: With the given parameters, we were able to achieve a time performance that we present, in the case of the argmin computation, and for every operation, in

table 5 in appendix B. We obtained these numbers by running the operations on an Intel Core i7-6600U CPU. In the following, when we talk of a "classification time", we mean the process of finding the closest encrypted embedding to the clear one. We therefore do not include the time that it takes to obtain the embedding from the audio sample through the VGGVox CNNs.
As for the distance computation, one homomorphic computation of equation 1 with the SEAL library takes less than 1e−4 seconds. Using the matrix version of the distance computation (and actually computing the difference in distances) as shown in equation 2 does not impact that timing significantly.
The overall time for a classification (the distance computations and the argmin computation) depends on the number of original model embeddings to which we need to compare the new input. To be more precise, for a number d of models, the tree scheme will evaluate $d-1$ comparison circuits (as seen in figure 3). This means the time the tree version takes for a classification is linear in the number of model embeddings.
As for the matrix scheme, it will first evaluate $(\frac{d^2-d}{2})$ BootStrap operations, and then, depending on the number of δ values that it can sum together at once before a BootStrap operation, it will evaluate a varying number of BootStrap operations. Most importantly, the time the matrix version takes for a classification is quadratic in the number of models.
Asymptotically speaking, the tree version is therefore better. This makes up for the fact that it is levelled and not fully homomorphic. However, given the weight of the CircuitBoot operation (see table 5) the tree version is actually significantly slower for a "low" number of models. As we can see in figure 5, "low" here means roughly under 4000 models. Under that threshold, the matrix scheme is more time-efficient. Figure 6 shows the time that the matrix version takes to homomorphically classify an input for up to 300 model embeddings. It takes less than a second to classify the speaker of a sample as one of 100 hidden speakers. We show in figure 7 the time that an argmin computation takes with the matrix version (the classification without the distance computations) depending on the number of values to compare. We can see that it takes less than a second to determine the argmin of 330 different encrypted values. As we have seen, solving the k-NN problem for up to 65 encrypted inputs with our scheme is as quick as solving the 1-NN problem. We can do this in less than 0.4 seconds.

6 Conclusion

In this paper, we investigated two different methods to classify a clear recording as coming from one of an arbitrary number of encrypted speaker embeddings. One such method is fully homomorphic but has a quadratic time-wise complexity in the number of models to classify from. The second one is levelled but time-linear in the number of models. Additionally, both of these method use their own way to compute a min and argmin from an arbitrary number of encrypted inputs using FHE and LHE respectively. Finally, a fully homomorphic and highly efficient k-NN computation was provided on top the previous results, and though it was not used in our classification work and is structurally limited in the number of inputs it can take.

Throughout this paper, we have mentioned promising prospects for future improvements of our results. Among those, a very efficient way to transform our levelled scheme by using a different type of MUX gate if the precision of the TFHE library increases. Additionally, it is paramount to stress the general applicability of our results to any embedding-based Euclidean classification system. The fact that we provide an efficient embedding-based classification here therefore expands the reach of homomorphic secure classification beyond this specific speaker-recognition system without needing to adapt any of our results to the specifics of the new system.

References

1. Albrecht, M., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Mathematical Cryptology, ePrint Archive 2015/046 (2015)
2. Ball, M., Carmer, B., Malkin, T., Rosulek, M., Schimanski, N.: Garbled neural networks are practical. Cryptology ePrint Archive, Report 2019/338 (2019)
3. Bergamaschi, F., Halevi, S., Halevi, T., Hunt, H.: Homomorphic Training of 30,000 Logistic Regression Models (2019)
4. Boura, C., Gama, N., Georgieva, M.: Chimera: a unified framework for b/fv, tfhe and heaan fully homomorphic encryption and predictions for deep learning. Cryptology ePrint Archive, Report 2018/758 (2018)
5. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Proceedings of CRYPTO 2018. Springer (2018)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277 (2011)
7. Carpov, S., Gama, N., Georgieva, M., Troncoso-Pastoriza, J.R.: Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption. Cryptology ePrint Archive, Report 2019/101 (2019)
8. Chabanne, H., Lescuyer, R., Milgram, J., Morel, C., Prouff, E.: Recognition Over Encrypted Faces: 4th International Conference, MSPN 2018, Paris, France (2019)
9. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035 (2017)
10. Chen, H., Chillotti, I., Dong, Y., Poburinnaya, O., Razenshteyn, I., Riazi, M.S.: Sanns: Scaling up secure approximate k-nearest neighbors search. Cryptology ePrint Archive, Report 2019/359 (2019)
11. Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library - seal v2.1 (2017)
12. Cheon, J.H., Kim, D., Park, J.H.: Towards a practical clustering analysis over encrypted data. IACR Cryptology ePrint Archive (2019)
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870 (2016)
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping. IACR Cryptology ePrint Archive p. 430 (2017)
15. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library (August 2016), https://tfhe.github.io/tfhe/
16. Chung, J.S., Nagrani, A., Zisserman, A.: Voxceleb2: Deep speaker recognition (2018)
17. Chung, J.S., Nagrani, A., Zisserman, A.: Voxceleb2: Deep speaker recognition. CoRR (2018)

18. Demmler, D., Schneider, T., Zohner, M.: Aby - a framework for efficient mixed-protocol secure two-party computation (2015)
19. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) Privacy Enhancing Technologies. Springer, Berlin and Heidelberg (2009)
20. Failla, P., Barni, M., Catalano, D., di Raimondo, M., Labati, R., Bianchi, T.: Privacy-preserving fingercode authentication (2010)
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012)
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. STOC '09 (2009)
23. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Cryptology ePrint Archive, Report 2013/340 (2013)
24. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
25. Halevi, S., Shoup, V.: Algorithms in helib. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. Springer, Berlin and Heidelberg (2014)
26. Huang, Y., Malka, L., Evans, D., Katz, J.: Efficient privacy-preserving biometric identification. In: NDSS (2011)
27. Jäschke, A., Armknecht, F.: Unsupervised machine learning on encrypted data. IACR Cryptology ePrint Archive (2018)
28. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X.: Secure logistic regression based on homomorphic encryption: Design and evaluation. In: JMIR medical informatics (2018)
29. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. Springer (2010)
30. Nagrani, A., Chung, J.S., Zisserman, A.: Voxceleb: a large-scale speaker identification dataset (2017)
31. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing. ACM (2005)
32. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press pp. 169–179 (1978)
33. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: Deepsecure: Scalable provably-secure deep learning. CoRR (2017)
34. Sadeghi, A.R., Schneider, T., Wehrenberg, I.: Efficient privacy-preserving face recognition. vol. Vol. 5984, pp. 229–244 (2009)
35. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. CoRR (2015)
36. Microsoft SEAL (release 3.2). https://github.com/Microsoft/SEAL (Feb 2019), microsoft Research, Redmond, WA.
37. Shaul, H., Feldman, D., Rus, D.: Scalable secure computation of statistical functions with applications to k-nearest neighbors. CoRR (2018)
38. Xie, P., Bilenko, M., Finley, T., Gilad-Bachrach, R., Lauter, K.E., Naehrig, M.: Crypto-nets: Neural networks over encrypted data. CoRR (2014)
39. Yao, A.C.C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science. SFCS '86, IEEE Computer Society (1986)

A Parameters

In this appendix we present the security parameters we use for both FV and TFHE.

λ	q	N	σ
80	132120577	1024	2e−11
110	132120577	1024	9e−11

λ	N	σ
80	1024	5e−13
80	2048	2e−25
110	1024	1e−9
110	2048	5e−19

Table 2. Tables presenting the secuirty parameters for both FV and TFHE. In the left table, the security parameters for FV. In the right table, the security parameters for TFHE. σ is the standard deviation of the Gaussian noise. Because we use both FV and TFHE, we need to choose the tighter security constraint between the two. The actual standard deviations chosen in the case of $N=1024$ for both 80 and 110 security levels are therefore written in green in the tables. The ones we cannot chose are in red.

N	p	q	σ
1024	10752	132120577	7.6e-9

Table 3. The parameters set in the SEAL library for the initial distance computations. N is the size of the initial ciphertext polynomials.

N_b	σ_b	B_g	ℓ
2048	1.1e-16	64	6

N_b	σ_b	B_g	ℓ	N_s	σ_s	base	t
2048	1.1e-16	16	10	1024	5e-13	5	14

Table 4. Tables presenting the appropriate parameters for both the matrix and tree schemes. parameters In the upper table, the parameters set in the case of the "matrix" argmin computation. In the lower table, the parameters set in the case of the "tree" scheme. N_b (resp. N_s) is the size of the bootstrapping key (resp. the key-switching key) polynomials and σ_b (resp. σ_s) the standard deviation of the Gaussian noise in the bootstrapping key (resp. the key-switching key).

B Timing figures

In this appendix we present table 5 and figures 5 6 and 7, all relating to the timing of the tree and matrix classification times as well as of the argmin problem resolution on its own.

"tree" BootStrap	CircuitBoot	MUX	KeySwitch
2e−5	0.4	2e−4	0.02

"matrix" BootStrap	InternAdd
1e−5	8e−7

Table 5. The average time for every basic operation used both in the "tree" scheme (upper table) and the "matrix one" (lower table). Times are given in seconds. In order to not burden the reader with too many timing numbers, these are the most significant operations. For instance, a Rotate operation time is insignificant in comparison, even taking into account the number of times that it is used.

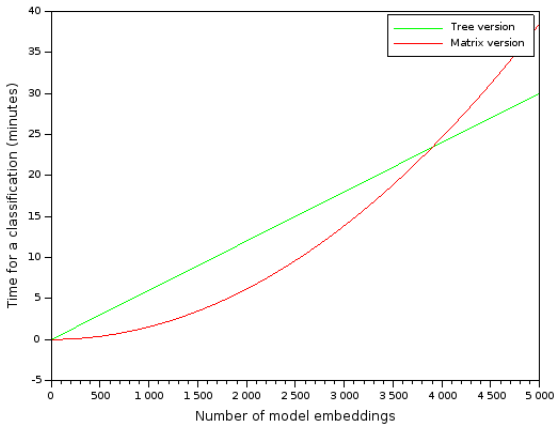


Fig. 5. Time in minutes for tree and matrix overall classification times depending on the number of model embeddings for the VGGVox system. The time for the tree version corresponds to the green curve and the matrix version to the red curve.

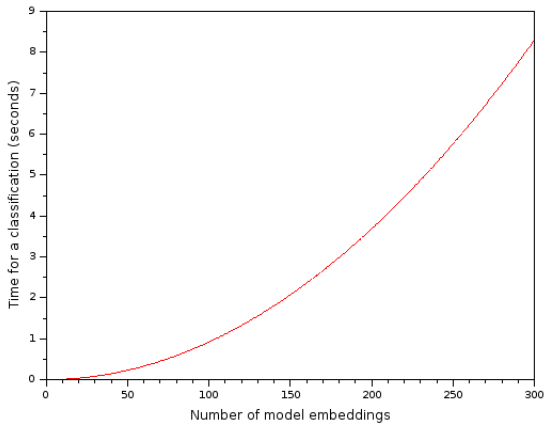


Fig. 6. Time in seconds for the matrix version overall classification times depending on the number of model embeddings for the VGGVox system.

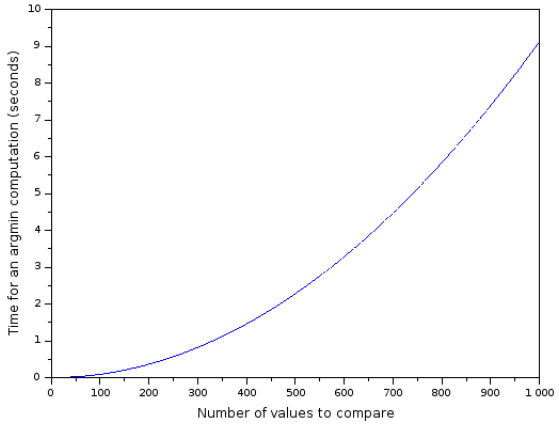


Fig. 7. Time in seconds for the argmin computation time with the matrix version depending on the number of vectors to compare.