Building blocks for LSTM homomorphic evaluation with TFHE

Daphne Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey

Building blocks for LSTM homomorphic evaluation with TFHE*

Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey

Université Paris-Saclay, CEA-List, Palaiseau, France
daphne.trama@cea.fr
aymen.boudguiga@cea.fr
pierre-emmanuel.clet@cea.fr
renaud.sirdey@cea.fr

Abstract. Long Short-Term Memory (LSTM) is a Neural Network (NN) type that creates temporal connections between its nodes. It models sequence data for applications such as speech recognition, image captioning, DNA sequence analysis, and sentence translation. Applications that are often subject to privacy constraints. This paper thus presents basic building blocks for the homomorphic execution of an LSTM that would respect the privacy of its inputs. By means of TFHE functional bootstrapping, we propose several approaches for homomorphically evaluating discretized flavors of the Sigmoid and Tanh activation functions. Experimental results show that the accuracy of the resulting discretized networks remains comparable to a full precision clear-domain execution. Performance-wise, we are able to homomorphically compute a Sigmoid or Tanh function in 0.3 or 0.15 secs (depending on whether or not multivalue bootstrapping is relied on). This paves the way towards evaluating practical LSTMs over encrypted inputs in around 1 to 3 mins which is competitive with the state of the art.

Keywords: LSTM · Privacy · TFHE · Homomorphic encryption.

1 Introduction

In recent years, Artificial Intelligence techniques and particularly Neural Networks (NN) have become pervasive in our connected society. They have already led to countless practical applications impacting, for better or worse, our daily lives. Examples are numerous: image recognition [23], artwork [27], interactive chat [25], voice analysis [28], and music generation [15]. In this context, protecting user data privacy during the operational phase of already trained neural nets has become a major challenge. In the field of provably-secure cryptography, Fully Homomorphic Encryption (FHE) is one major corpus of techniques serving as a yardstick to address such privacy challenges.

This paper thus focuses on running a special type of neural network, called Recurrent Neural Networks (RNN), over encrypted inputs. These networks rely, in particular, on a specific type of unit called Long-Short Term Memory (LSTM), first

* This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute.

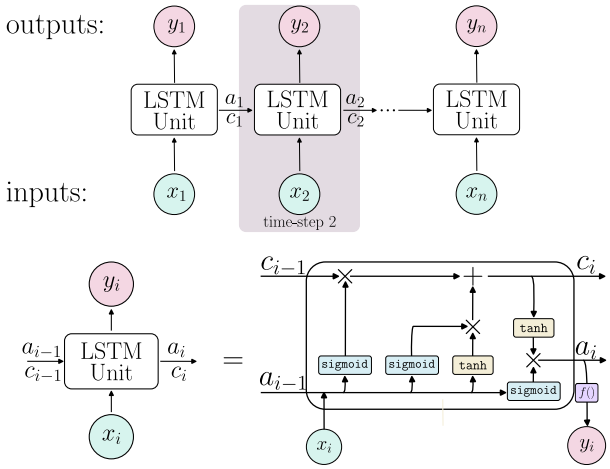

Fig. 1: LSTMs structure: the LSTM unit of time-step i takes as input x_i as well as the outputs of the LSTM unit at time-step $i-1$, namely the activation a_{i-1} and the memory cell c_{i-1}. It then outputs $y_i = f(a_i)$, where f can be an activation function or simply the identity function.

proposed in 1997 in [18]. The main idea of LSTM is to use memory cells to capture the relation with activations from previous time-steps (as described in Fig. 1). LSTM relies on Sigmoid and Tanh for memory cells and activation computation. These networks are mainly used for text translation, natural language processing, language translation, feeling analysis, and speech analysis.

Still, in these kinds of applications, input confidentiality can be crucial (and so is the confidentiality of any byproduct of these inputs). For instance, it is generally the case when the network manipulates medical data sent for diagnosis findings, genomic data, or sensitive voice data. Furthermore, users (e.g., patients, hospitals, or governments) may wish to prevent neural network owners from accessing their sensitive data and extracting information from it as a byproduct. It is therefore desirable to encrypt these input data and then, as a consequence, evaluate the full neural network in the encrypted domain. This eventually leads to encrypted results which are sent back to the user who, alone, should be granted access to them. Thus, it becomes possible to delegate classification tasks to third parties while protecting the confidentiality of the data. Utopian? Not at all. Indeed, with respect to Convolutional Neural Networks (CNN), starting with the work of Dowlin *et al.* on CryptoNets [14], several works have investigated the application of various flavors of homomorphic encryption techniques to various flavors of neural networks with the long-term goal of achieving the above setup [7, 6, 21, 2, 19, 24, 1, 10]. However, these works have so far achieved limited scaling (reaching throughputs ranging from a few hundred to a few thousand neurons per minute) and had to resort to simple activation functions (for instance square, Sign, or ReLU).

In this context, this paper further investigates how the Sigmoid (and hyperbolic tangent Tanh) function can be evaluated in the homomorphic domain, particularly, to run RNNs. On the one hand, RNNs are exciting candidates for FHE since their

applications often require confidentiality. On the other hand, contrary to the more classical CNN, RNN experimentally appears much more sensitive to issues relative to discretization and activation function precision. Nevertheless, the other operations necessary for running the network, such as additions or scalar products, have already been studied and are implemented using classical circuits for basis B arithmetic. But Sigmoid and Tanh still need to receive satisfactory treatment from a homomorphic encryption point of view (mainly because they have been worked around in previous studies focusing on CNN). Consequently, this paper primarily focuses on the Sigmoid and Tanh functions as "must-have" building blocks for running RNNs over homomorphically encrypted data. However, these non-linear functions must be approximated to fit the constraints of homomorphic encryption operations[1]. One common approach is to approximate them by polynomials [26, 20, 8], but this raises a number of issues. First, low-degree polynomial approximations are accurate around 0. Second, with leveled homomorphic schemes such as BFV [3, 16] or BGV [4], the multiplicative depth of an LSTM increases unboundedly with the number of successive time-steps because of the recurrent nature of these networks. Thus, even for small LSTMs, the leveled homomorphic scheme parameters would not allow an efficient execution. Meanwhile, for larger LSTMs, leveled homomorphic schemes lead to prohibitive costs and the use of a (practical) bootstrapping-able homomorphic encryption scheme such as TFHE [9] becomes mandatory. However, with TFHE, using activation functions that are approximated with polynomials is very time-consuming. Indeed, with TFHE, we decompose every large input into digits in a basis B before encryption, and we express multiplication and addition as circuits for basis B arithmetic where each unitary operation requires at least one programmable (or functional) bootstrapping.

State of the art– In 2022, Jang *et al.*, [20] implemented an RNN with Gated Recurrent Units[2] (GRUs) with an improved version of the CKKS homomorphic encryption scheme. They enhanced CKKS with the support of multivariate RLWE (m-RLWE) samples. As a result, they were able to encode a matrix of clear values into one plaintext by using batching (then encrypting the obtained multivariate polynomial in one ciphertext). They also provided an efficient algorithm for matrix-vector multiplication and approximated Tanh and Sigmoid with polynomials of degree 7. With all these building blocks, they were able to evaluate RNNs with GRUs over encrypted data (with comparable accuracies) for sequence copy (4 GRUs), regression adding (1 GRU), image classification (28 GRUs), and genomic sequence classification (40 GRUs). For the small RNNs with 1 to 4 GRUs, Jang *et al.*, did not rely on bootstrapping, but they did need a bootstrapping per time-step for larger sequences– 28 to 40 GRUs (starting from the 4^{th} time-step). During their experiments, they compute each unit in 90 secs (without including the CKKS bootstrapping time which can range from 3 to 26 minutes depending on the number of used threads and with respect to the selected parameters in [20]).

[1] Note that, in principle, floating point functions can be performed by means of homomorphic computations (e.g. by running their boolean circuit representations over an FHE with $\mathbb{Z}_2$ as plain domain). In practice, however, such an approach induces prohibitive computational costs.

[2] GRU units are a simple version of LSTM ones. They also rely on Tanh and Sigmoid for computing memory cells and activations. However, they cannot manage very long dependencies.

Paul *et al.*, [26] proposed a collaborative training between two parties that share the same LSTM in clear form but intend to train an extra logistic regression layer. The authors used homomorphic encryption only for training the logistic regression layer. They confirmed that training and running the LSTM over encrypted data was complex and time-consuming.

Contribution– In this paper, we design Look-Up Tables (LUTs) for evaluating Tanh and Sigmoid using TFHE functional bootstrapping. In order to do so, we propose several approaches to homomorphically evaluate discretized variants of the Sigmoid and Tanh activation functions. We then introduce two step-wise functions as replacements for Sigmoid and Tanh, which we experimentally show are sufficient to preserve the accuracy of a real-world RNN. In terms of performances, we are then able to homomorphically compute a Sigmoid or a Tanh in less than half a second, which is competitive with the state of the art.

Paper organization– This paper is organized as follows: Section 2 reviews the basics of the TFHE cryptosystem and gives the necessary details about LSTMs structure to render this paper self-contained. In Section 3 we discuss a number of discretization issues in LSTM. In particular, we evaluate the impact on the accuracy of the network when Sigmoid and Tanh are approximated by different discrete functions. Section 4 provides a detailed exposition of our approaches to executing our approximated activation functions in the FHE domain. We also present the performances of our methods. Section 5 concludes the paper and gives some perspectives.

2 Preliminaries

2.1 Notations

Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ be the real torus, that is to say, the additive group of real numbers modulo 1 ($\mathbb{R} \mod 1$). We will denote by $\mathbb{T}_N[X]^n$ the set of vectors of size n whose coefficients are polynomials of $\mathbb{T} \mod (X^N + 1)$. N is usually a power of 2. Let $\mathbb{B} = \{0,1\}$. $\langle \, , \, \rangle$ denotes the inner product.

2.2 TFHE Scheme

The TFHE scheme is a fully homomorphic encryption scheme introduced in 2016 in [9] and implemented as the TFHE library [3]. TFHE defines three structures to encrypt plaintexts, which we summarize below as fresh encryptions of 0:

- **TLWE sample**: A pair $(a,b) \in \mathbb{T}^{n+1}$, where a is uniformly sampled from $\mathbb{T}^n$ and $b = \langle a,s \rangle + e$. The secret key s is uniformly sampled from $\mathbb{B}^n$, and the error $e \in \mathbb{T}$ is sampled from a Gaussian distribution with mean 0 and standard deviation σ.
- **TRLWE sample**: A pair $(a,b) \in \mathbb{T}_N[X]^{k+1}$, where a is uniformly sampled from $\mathbb{T}_N[X]^k$ and $b = \langle a,s \rangle + e$. The secret key s is uniformly sampled from $\mathbb{B}_N[X]^k$, the error $e \in \mathbb{T}$ is a polynomial with random coefficients sampled from a Gaussian distribution with mean 0 and standard deviation σ. One usually chooses $k = 1$, therefore a and b are vectors of size 1 whose coefficient is a polynomial.

[3] https://tfhe.github.io/tfhe/

– **TRGSW sample**: a vector of l TRLWE fresh samples.

Let $\mathcal{M}$ denote the discrete message space ($\mathcal{M} \in \mathbb{T}_N[X]$ or $\mathcal{M} \in \mathbb{T}$). To encrypt a message $m \in \mathcal{M}$, we add what is called a *noiseless trivial* ciphertext $(0,m)$ to a fresh encryption of 0. We denote by $c=(a,b)+(0,m)=(a,b+m) \in \mathrm{T(R)LWE}_s(m)$ the T(R)LWE encryption of m with key s. A message $m \in \mathbb{T}_N[X]$ can also be encrypted in TRGSW samples by adding $m \cdot H$ to a TRGSW sample of 0, where H is a gadget decomposition matrix. As we will not implicitly need such an operation in this paper, more details about TRGSW can be found in [9].

To decrypt a ciphertext c, we first calculate its phase $\phi(c) = b - \langle a,s \rangle = m + e$. Then, we need to remove the error, which is achieved by rounding the phase to the nearest valid value in $\mathcal{M}$. This procedure fails if the error is greater than half the distance between two elements of $\mathcal{M}$.

2.3 TFHE Bootstrapping

Bootstrapping is the operation that reduces the noise of a ciphertext thus allowing further homomorphic calculations. It relies on three basic operations, which we briefly review in this section.

– **BlindRotate**: rotates a polynomial encrypted as a TRLWE ciphertext by an encrypted index. It takes several inputs: a ciphertext $c \in \mathrm{TRLWE}_k(m)$, a vector $(a_1,\cdots,a_p,b)$ where $\forall i, a_i \in \mathbb{Z}_{2N}$, and a TRGSW ciphertext encrypting the secret key $s=(s_1,\cdots,s_p)$.
It returns a ciphertext $c' \in \mathrm{TRLWE}_k(m \cdot X^{\langle a,s \rangle - b})$. In this paper, we will refer to this algorithm as `BlindRotate`.
– **TLWE Sample Extract**: extracts a coefficient of a TRLWE ciphertext and converts it into a TLWE ciphertext. It takes as inputs both a ciphertext $c \in \mathrm{TRLWE}_k(m)$ and an index $p \in \{0, \cdots, N-1\}$. The result is a TLWE ciphertext $c' \in \mathrm{TLWE}_k(m_i)$ where m_i is the i^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm as `SampleExtract`.
– **Public Functionnal Keyswitching**: allows the switching of keys and parameters from p ciphertexts $c_i \in \mathrm{TLWE}_k(m_i)$ to one $c' \in \mathrm{T(R)LWE}_s(f(m_1,\cdots,m_p))$ where f is a public linear morphism between $\mathbb{T}^p$ and $\mathbb{T}_N[X]$. That is to say, this operation not only allows the packing of TLWE ciphertexts in a TRLWE ciphertext, but it can also evaluate a linear function f over the input TLWEs. In this paper, we will refer to this algorithm as `KeySwitch`.

It is important to note that, during a `BlindRotate` operation, an excessive noise level in the input ciphertext can lead to errors in the bootstrapping output resulting in incorrect ciphertexts (i.e. ciphertext which do not decrypt to correct calculation results). This has implications for parameters and data representations choices (number of digits and basis), as we shall later see in Sect. 4.

Algorithm 1 shows the *TFHE Gate Bootstrapping* [9], which aims to evaluate a binary gate operation homomorphically and reduce the output ciphertext noise at the same time. To that end, 0 and 1 are respectively encoded as 0 and $\frac{1}{2}$ over

$\mathbb{T}$. The first step of this algorithm consists in selecting a value $\hat{m} \in \mathbb{T}$ which will be used afterward to compute the coefficients of the polynomial which will rotate during the `BlindRotate`. We call this polynomial *testv* as seen in Step 3. Note that for any $p \in [\![0,2N]\!]$ (where $[\![0,2N]\!]$ corresponds to the set of integers $\{0,\cdots,2N\}$), the constant term of $testv \cdot X^p$ is $\hat{m}$ if $p \in]\!]\frac{N}{2},\frac{3N}{2}[\![$ and $-\hat{m}$ otherwise. Step 4 returns an accumulator $ACC \in \mathrm{TRLWE}_{s'}(testv \cdot X^{\langle \overline{a},s \rangle - b})$. Indeed, the constant term of ACC is $-\hat{m}$ if c is an encryption of 0 and $\hat{m}$ if c is an encryption of $\frac{1}{2}$. Then step 5 creates a new ciphertext $\overline{c}$ by extracting the constant term in position 0 from ACC and adding $(0,\hat{m})$. Thus, $\overline{c}$ corresponds to an encryption of 0 if c is an encryption of 0 and m otherwise. On the other hand, if c is an encryption of $\frac{1}{2}$ and if we choose $m = \frac{1}{2}$, the algorithm returns a *fresh sample* of $\frac{1}{2}$, that is to say the encoding of 1.

In Fig. 2, we present an example of TFHE gate bootstrapping algorithm with $\mathbb{Z}_4 = \{0,1,2,3\}$ as input space. The outer circle in Fig. 2 corresponds to the plaintext encoding in $\mathbb{T}$ as $\{0,\frac{1}{4},\frac{2}{4},\frac{3}{4}\}$. Meanwhile, the inner circle sets the coefficients of the test polynomial *testv* to 1, i.e., $\hat{m} = \frac{1}{4}$. Then, we rotate the test polynomial during the bootstrapping by the phase $\phi(c_0)$ of the input ciphertext c_0. In our example, we obtain as bootstrapping output either an encryption of the encoding of 1 for positive inputs $\{0,\frac{1}{4}\}$, or an encryption of the encoding of -1 for negative inputs $\{\frac{2}{4},\frac{3}{4}\}$.

Algorithm 1 TFHE gate bootstrapping [9]

Require: a constant $m \in \mathbb{T}$, a TLWE sample $c = (a,b) \in \mathrm{TLWE}_s(x \cdot \frac{1}{2})$ with $x \in \mathbb{B}$, a bootstrapping key $BK_{s \to s'} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key s'.

Ensure: a TLWE sample $\overline{c} \in \mathrm{TLWE}_s(x.m)$

1: Let $\hat{m} = \frac{1}{2}m \in \mathbb{T}$ (pick one of the two possible values)
2: Let $\overline{b} = \lfloor 2Nb \rceil$ and $\overline{a_i} = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
3: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \hat{m} \in \mathbb{T}_N[X]$
4: $ACC \leftarrow$ `BlindRotate`$((0,testv),(\overline{a}_1,...,\overline{a}_n,\overline{b}),(BK_1,...,BK_n))$
5: $\overline{c} = (0,\hat{m}) +$ `SampleExtract`(ACC)
6: return `KeySwitch`$_{s' \to s}(\overline{c})$

2.4 TFHE Functional Bootstrapping

We can use Look-Up Tables to compute functions during the bootstrapping operation. To do so, we replace the coefficients of the test polynomial *testv* with the corresponding values of the LUT. Let us assume that we want to evaluate the function $f_{\mathbb{T}}$ via a LUT. Then, if we retrieve the i^{th} coefficient of *testv*, we actually get $f_{\mathbb{T}}(m_i)$ where m_i is the encrypted input to the bootstrapping. We refer to this idea by *programmable* or *functional* bootstrapping [11, 22, 30, 12, 13].

In Fig. 2, we give an example of functional bootstrapping with $\mathbb{Z}_4 = \{0,1,2,3\}$ as input space. We encode the images of $\{0,\frac{1}{4}\}$ by $f_{\mathbb{T}}$ as coefficients of the test polynomial (in the inner circle). Meanwhile, we deduce the images of $\{\frac{2}{4},\frac{3}{4}\}$ by negacyclicity. Indeed, in $\mathbb{T}$, we can encode negacyclic functions i.e., antiperiodic functions with

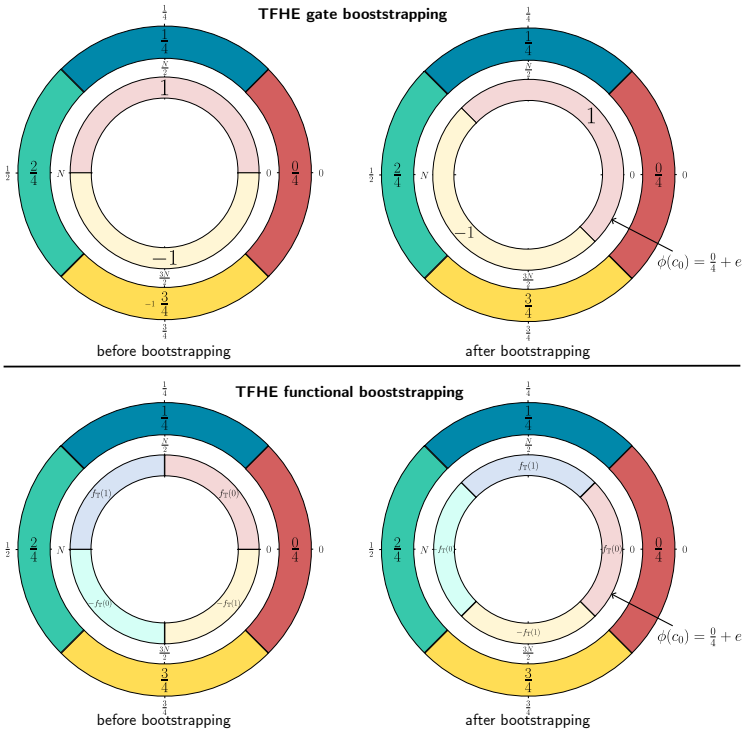

Fig. 2: TFHE Bootstrapping examples: the outer circles describe the inputs to the bootstrapping (i.e., ciphertexts over $\mathbb{T}$). Meanwhile, the inner circles represent the coefficients of the test polynomial *testv*. One of these coefficients is extracted as the output of the bootstrapping after the `BlindRotate`.

period $\frac{1}{2}$ (verifying $f_{\mathbb{T}}(x) = -f_{\mathbb{T}}(x+\frac{1}{2})$), where [0,0.5[corresponds to positive values and [0.5,1[to negative ones. In our example, if we encrypt one of the following values $\{0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}\}$ and we give it as input to the functional bootstrapping algorithm, we get $\{f_{\mathbb{T}}(0), f_{\mathbb{T}}(1), -f_{\mathbb{T}}(0), -f_{\mathbb{T}}(1)\}$, respectively.

Almost all of the functional bootstrapping methods from state of the art ([11, 22, 30, 12, 13]) take as input a single ciphertext. In 2021, Guimarães *et al.*, [17] discussed two methods for performing functional bootstrapping with larger plaintexts. They combine several bootstrappings with different encrypted inputs by using a tree or a chain structure. The ciphertexts are encryptions of digits that come from the decomposition of plaintexts in a certain basis B.

In this paper, we will instantiate Guimarães *et al.* tree method to perform a homomorphic evaluation of the functions Sigmoid and Tanh or, rather, suitable approximations of them.

2.5 Long Short Term Memory (LSTM)

This section reviews the equations that govern LSTMs, highlighting the recurrent aspect of the computations to be performed. We denote by $c^{\langle i \rangle}$ the memory variable of the i^{th} cell. $\tilde{c}^{\langle i \rangle}$ is the candidate value to update the memory variable. We respectively write Γ_u for the update gate, Γ_f for the forget gate and Γ_o for the output gate. We denote by W_j a weight matrix and by b_j the associate bias. x is the input and σ stands for the Sigmoid. Finally, we give the equations that define the i^{th} LSTM unit:

$$
\begin{aligned}
\tilde{c}^{\langle i \rangle} &= \tanh(W_c \cdot [c^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_c) \\
\Gamma_u^{\langle i \rangle} &= \sigma(W_u \cdot [c^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_u) \\
\Gamma_f^{\langle i \rangle} &= \sigma(W_f \cdot [c^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_f) \\
\Gamma_o^{\langle i \rangle} &= \sigma(W_o \cdot [c^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_o) \\
c^{\langle i \rangle} &= \Gamma_u \cdot \tilde{c}^{\langle i \rangle} + (1 - \Gamma_u) \cdot c^{\langle i-1 \rangle} \\
a^{\langle i \rangle} &= \tanh(W_a \cdot [a^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_a)
\end{aligned}
$$

To perform a confidentiality-preserving LSTM evaluation, all these different operations must be done in the homomorphic domain. In this paper, as a necessary first step towards running RNNs over FHE, we focus mainly on the two building blocks that are Sigmoid and Tanh as they are in need to receive satisfactory treatment from a homomorphic encryption point of view.

3 Discretization Issues in LSTM

In this work, we aim at running LSTM building blocks with TFHE functional bootstrapping. First, we decompose our input data into a certain basis B before their encryption (as TFHE functional bootstrapping does not support large plaintexts [13]). Then, we discretize our functions to encode them as a LUT in the test polynomial *testv*. We propose to work with a practical use case that allows studying the loss of accuracy due to discretization.

3.1 Use Case

We choose to work on the LSTM proposed by J. Woodbridge's team in [29]. Indeed, their LSTM takes as input a domain name and determines whether it is a malicious domain name, which seems appropriate to our study because the LSTM is quite long and represents real-world usage.

The network is composed of several layers: an embedding layer, an LSTM layer, and a fully connected layer. The LSTM layer is composed of 128 LSTM units, which we discretized to support an evaluation with homomorphic encrypted inputs. To do so, we successfully reproduced their experimental results and computed the accuracy of the network: 95.6%. We use this accuracy value as a reference when experimenting with our discretized variants.

3.2 Coping with TFHE Clear Domain Size Constraints

The first step to make the network homomorphic consists in discretizing the different data. Indeed, let us recall that the ciphertext domain $\mathcal{M}$ in TFHE is discrete, generally of size $p=16$ or $p=32$ whereas the inputs of the LSTM layer usually are floating-point numbers. For a fully homomorphic evaluation of the network, these inputs must be encrypted, hence discretized on p values. In our case, it means selecting the p most relevant floating-point values and assigning these values as *interpretations* of $\mathcal{M} = \{0, \frac{1}{p}, \frac{2}{p}, \cdots, \frac{p-1}{p}\}$. To put it another way, one has to choose how to encode p values of $\mathbb{R}$ into $\mathcal{M}$. So we have to find a bijection $\iota : F \to \mathcal{M}$, where F is a set of p floating-point numbers. It is the values of F that we have to determine adequately. Here "adequately" means that by replacing the inputs (recall that in our network they correspond to the output of an embedding layer) with the values found for F, the network keeps the same predictions as on the original inputs. To do this, we need to start by looking at what the inputs of the LSTM layer look like in typical execution. Thus, the coefficients of the network inputs are all between -1 and 1. So none of the p values will be chosen outside $[-1,1]$. Looking further, we notice that their distribution is not uniform. So choosing $F = \{-1 + \frac{2i}{p} \mid i \in \{0,\cdots,p-1\}\}$ is not appropriate. We then thought of the k-means clustering method.

K-means clustering is a vector quantization method that aims to partition n elements into k clusters in which each element belongs to the cluster with the nearest mean. We used this method reduced to one dimension on our set of inputs in order to determine p clusters. We then pick out the center of each cluster to obtain our set F. We replace each input coefficient with the value of its cluster center in F and run the tests. As we observe that the resulting network accuracy does not change with such a strong discretization, we then apply the same approach to the coefficients of the weight matrices and bias vectors. Again, despite such a drastic data discretization, the tests performed on the new discretized network show that these simplifications do not influence the final accuracy. Indeed, the discretized network also achieves 95.6% of correct predictions. The discretization prerequisites towards a homomorphic execution, therefore, appear to be met.

3.3 A Naive Discretization of the Activation Functions

As it stands, TFHE does not allow, even with programmable bootstrapping, to practically evaluate functions such as Sigmoid or Tanh. Indeed, efficiently evaluating non-linear functions with high precision remains a challenge for FHE schemes. A naive approach is then to coarsely approximate the activation functions by much simpler step functions. Such approaches have already been carried out on CNNs [2], with encouraging results, but never on RNNs. It is thus legitimate to start by trying a similar approach for LSTM.

For the Sigmoid which has values in $]0,1[$, we can choose the Heaviside function, while for the Tanh with values in $]-1,1[$, we can choose the Sign function. These are (very) rough approximations, with two steps, but the question is whether the network can endure such a simplification of its activation functions. We first perform discretization tests with clear data with these naive approximations. Contrary to the

CNN case, the results proved to be negatively conclusive: discretizing the activation functions in only two steps leads to a significant loss of accuracy. Indeed, as shown in the first rows of Table 1, the network accuracy drops to 0.50.

3.4 A Finer Stepwise Approach

More accurate approximations can be obtained by means of stepwise approximation as initially suggested by Guimarães *et al.* [17] who presented a 3-steps approximation of the Sigmoid for illustrative purpose and gave hints at how to implement it by means of functional bootstrapping. We call these variants 3StepsSigmoid and 3StepsTanh.

Thus, we first considered this approximation but the results, shown in Table 1, although better than those obtained in the previous section, lead to an unacceptable 10 points loss in network accuracy. We thus investigated a more precise approximation of our own with two steps on the interval $]-\infty,-1]$, an affine part on $]-1,1]$, and two more steps on $]-1,\infty[$. This leads to the following functions, corresponding to the curves presented in Fig. 3.

$$\begin{aligned}\mathsf{StepSigmoid}(x) = {} & 0.13\cdot\mathbb{1}_{]-6,-1]}(x) \;+\; \mathbb{1}_{]-1,1]}(x)\cdot(0.24x+0.5) \;+\; 0.87\cdot\mathbb{1}_{]1,6]}(x) \\ & +\; \cdot\mathbb{1}_{]6,\infty[}(x) \\ \mathsf{StepTanh}(x) = {} & -1\cdot\mathbb{1}_{]-\infty,-3]}(x) \;-\; 0.875\cdot\mathbb{1}_{]-3,-1]}(x) \;+\; \mathbb{1}_{]-1,1]}(x)\cdot(0.76x) \\ & +\; 0.92\cdot\mathbb{1}_{]1,6]}(x) \;+\; \cdot\mathbb{1}_{]6,\infty[}(x).\end{aligned}$$

where $\mathbb{1}_F$ is the indicator function and F is a subset of a set E.

$$\begin{aligned}\mathbb{1}_F : E &\to \quad \{0,1\} \\ x &\mapsto \begin{cases} 1 \text{ if } x \in F \\ 0 \text{ otherwise.} \end{cases}\end{aligned}$$

Again, to determine the optimized values for the steps of our functions, we used

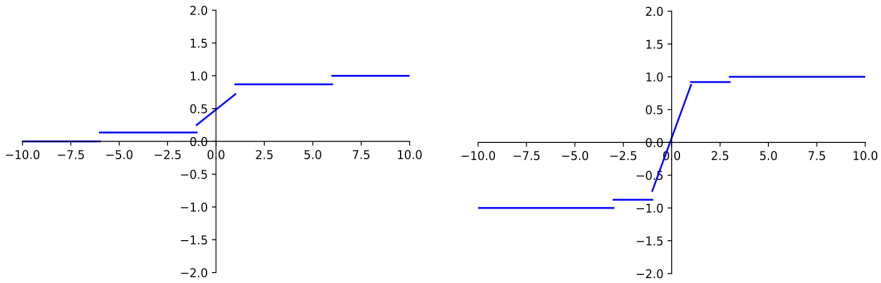

Fig. 3: Our stepwise activation functions StepSigmoid and StepTanh

the k-means clustering method (as in Sect. 3.3) on the clear outputs of Sigmoid and Tanh, leading us to an optimized discretized version of the activation functions. Using these two approximations to replace the Sigmoid and Tanh functions, we obtain a final accuracy of the network of 93.4%, only 2% below the network accuracy under floating-point precision.

used functions	accuracy of the LSTM
Tanh + Sigmoid	0,956
Sign + Heaviside	0,50
3StepsTanh + 3StepsSigmoid [17]	0,832
StepTanh + StepSigmoid	0,934

Table 1: Accuracy of the network depending on the activation functions

Lastly, to allow a homomorphic evaluation under TFHE, it is also necessary to discretize the affine part of our StepSigmoid and StepTanh in 12 intermediate steps. Indeed, to minimize the degradation of the accuracy, it is necessary to have a significant number of steps between -1 and 1 because the prediction of the network often relies on this interval of values. Thus, we obtain activation functions composed of 16 steps, corresponding to the $p=16$ values of the ciphertext space $\mathcal{M}$. The final discretization is given in Table 2. The accuracy of the network remains unchanged, with a rate of 93.4% of correct predictions.

Thus with our proposed approximated functions StepSigmoid and StepTanh, we have a network for which we can investigate whether a homomorphic evaluation is doable.

4 FHE Execution

This section presents the tree-based method, first introduced in [17]. This approach uses the output of a LUT to construct a new LUT, and thus allows the evaluation of functions by means of multiple functional bootstrappings. Each of these bootstrapping has as input a ciphertext encrypting a plaintext in a basis B.

4.1 Instantiation of the Tree-based Method for the Sigmoid

We now attempt to homomorphically evaluate StepSigmoid and StepTanh by means of a tree-based bootstrapping method.

Let $B,B',d\in\mathbb{N}^*$ and m be an integer message. B and B' are the basis on which to decompose the message. We then have $m=\sum_{i=0}^{d-1} m_i B^i$, with $m_i\in[\![0,B-1]\!]$. From this decomposition, we obtain d TLWE encryptions $(c_0,c_1,\cdots,c_{d-1})$ of $(m_0,m_1,\cdots,m_{d-1})$ on half of the torus $\mathbb{T}$. We denote $f:[\![0,B-1]\!]^d\to[\![0,B'-1]\!]$ the target function and define g as the following bijection:

$$\begin{array}{rccc} g: & [\![0,B-1]\!]^d & \to & [\![0,B^d-1]\!] \\ & (a_0,a_1,\cdots,a_{d-1}) & \mapsto & \sum_{i=0}^{d-1} a_i\cdot B^i \end{array}$$

x	StepSigmoid(x)	x	StepTanh(x)
]-∞, -6]	0	]-∞, -3]	-1
]-6, -1]	0.13	]-3, -1]	-0.875
]-1, -0.8461]	0.19	]-1, -0.8461]	-0.74
]-0.8461, -0.6922]	0.24	]-0.8461, -0.6922]	-0.61
]-0.6922, -0.5383]	0.30	]-0.6922, -0.5383]	-0.47
]-0.5383, -0.3844]	0.36	]-0.5383, -0.3844]	-0.37
]-0.3844, -0.2305]	0.41	]-0.3844, -0.2305]	-0.20
]-0.2305, -0.0766]	0.47	]-0.2305, -0.0766]	-0.07
]-0.0766, 0.0772]	0.53	]-0.0766, 0.0772]	0.07
]0.0772, 0.2310]	0.56	]0.0772, 0.2310]	0.20
]0.2310, 0.3848]	0.64	]0.2310, 0.3848]	0.33
]0.3848, 0.5386]	0.7	]0.3848, 0.5386]	0.47
]0.5386, 0.6924]	0.76	]0.5386, 0.6924]	0.60
]0.6924, 0.8462]	0.81	]0.6924, 0.8462]	0.74
]0.8462, 1]	0.87	]0.8462, 1]	0.875
]1, 6]	0.93	]1, 3]	0.92
]6, +∞]	1	]3, +∞]	1

Table 2: Our 16-steps StepSigmoid and StepTanh.

We then encode a LUT for f under the form of B^{d-1} TRLWE ciphertexts. Each of these ciphertexts encodes a polynomial P_i such that:

$$P_i(X) = \sum_{j=0}^{B-1} \sum_{k=0}^{\frac{N}{B}-1} f \circ g^{-1}(j \cdot B^{d-1} + i) \cdot X^{j \cdot \frac{N}{B} + k}$$

Then, we apply the `BlindRotateAndExtract` (the `BlindRotate` directly followed by the `SampleExtract` in position 0) to each test polynomial $testv = \text{TRLWE}(P_i)$ with c_0 as a selector. We obtain B^{d-1} TLWE ciphertexts, each corresponding to the encryption of $f \circ g^{-1}(m_{d-1} \cdot B^{d-1} + i)$, for $i \in [\![0, B^{d-1} - 1]\!]$.

Finally, we use the `KeySwitch` operation from TLWE to TRLWE to gather them into B^{d-2} encrypted TRLWE, corresponding to the LUT of h, with:

$$\begin{array}{rccc} h: & [\![0,B-1]\!]^{d-1} & \to & [\![0,B'-1]\!] \\ & (a_0,a_1,\cdots,a_{d-2}) & \mapsto & f(a_0,a_1,\cdots,a_{d-2},m_{d-1}) \end{array}$$

We then repeat this operation, using the ciphertext c_i at step i, until we obtain a single TLWE ciphertext of $f(m_0, m_1, \cdots, m_{d-1})$. Note that the tree-based method must be run independently as many times as the number of digits in the output.

With our plaintext space of size $p = 16$, we opt for a decomposition in a basis $B = 4$, and so we get $d = 2$. That is, the input of our bootstrapping StepSigmoid corresponds to encryptions of two integers in $\mathbb{Z}_4$. With this setting, we get a better network accuracy (93.4%) than when using a single integer in basis 16 (90.1%) thanks to lower bootstrapping error rates for the same parameters set. Indeed, the values of the test polynomial *testv* are spread over several vectors rather than encoded into one, which

results in a lower noise tolerance from the `BlindRotate` (as discussed in Sect. 2.3). We choose to output one ciphertext in $\mathbb{Z}_{16}$ ($B'=16$): to avoid running the tree-based method multiple times. With a one-digit output, we thus need only a single execution.

4.2 Multi-value Bootstrapping

Multi-Value Bootstrapping (MVB) [5] refers to the method for evaluating k different LUTs on a single input with a single bootstrapping. MVB factors the test polynomial P_{f_i} associated with the function f_i into a product of two polynomials v_0 and v_i, where v_0 is a common factor to all P_{f_i}. In practice, we have:

$$(1+X+\cdots+X^{N-1})\cdot(1-X)\equiv 2 \bmod (X^N+1)$$

Now by writing P_{f_i} in the form $P_{f_i}=\sum_{j=0}^{N-1}\alpha_{i,j}X^j$ with $\alpha_{i,j}\in\mathbb{Z}$, we get from the previous equation:

$$\begin{aligned} P_{f_i} &= \frac{1}{2}\cdot(1+X+\cdots+X^{N-1})\cdot(1-X)\cdot P_{f_i} \bmod (X^N+1) \\ &= v_0\cdot v_i \bmod (X^N+1) \end{aligned}$$

where:

$$\begin{aligned} v_0 &= \frac{1}{2}\cdot(1+X+\cdots+X^{N-1}) \\ v_i &= \alpha_{i,0}+\alpha_{i,N-1}+(\alpha_{i,1}-\alpha_{i,0})\cdot X+\cdots+(\alpha_{i,N-1}-\alpha_{i,N-2})\cdot X^{N-1}. \end{aligned}$$

This factorization makes it possible to compute many LUTs using a unique bootstrapping. Indeed, it is enough to initialize the test polynomial *testv* with the value of v_0 during bootstrapping. Then, after the `BlindRotate` operation, one has to multiply the obtained ACC by each v_i corresponding to the LUT of f_i to get ACC_i.

This optimization reduces the number of bootstrapping required for an operation and, thus, the overall computation time. In our case, the MVB allows us to reduce the number of bootstrapping from 5 to 2 when evaluating either the Sigmoid or the Tanh.

4.3 Technical Setup

We did our experiments with the default parameters of TFHElib[4]. The librairy gives $N=1024$ and $n=630$ so that $a\in\mathbb{T}^{630}$ (TLWE) or $a\in\mathbb{T}_{1024}[X]$ (TRLWE). We run our code on a 4-core Intel Core i7-7600U 2.90GHz CPU (*with only one core activated*) and 16GiB total system memory with a Ubuntu 20.04.5 LTS server.

In Sect. 3, we successfully discretized Sigmoid into StepSigmoid by encoding the Sigmoid domain on 16 distinct values. In practice, we set $p=32$ to be able to encode these 16 values on the positive half of the torus (i.e. [0,0.5[). As such, we ensure that

[4] https://tfhe.github.io/tfhe/ (v1.0.1-36-gbc71bfa).

our plaintext space is included in [0,0.5[. So, we are no more limited by negacyclic constraints [13].

Again, our StepSigmoid returns 16 positive values: {0.53,0.56,0.64,0.7,0.76,0.81,0.87, 1.0,0.47,0.41,0.36,0.3,0.24,0.19,0.13,0} $\in [0,1]$. These values will compose the test polynomial *testv*. As we only work with the 16 values of the positive half of the torus, i.e., $\{\frac{0}{32},\cdots,\frac{15}{32}\}$ corresponding to values in $[0,0.5[$, we must find a way to also return values in $[0.5,1[$ to reach all the images of our StepSigmoid. To solve this issue, we divide by 2 the values of the original *testv*, which are all now ≤ 0.5: {0.265,0.28,0.32,0.35,0.38,0.405,0.435,0.5,0.235,0.205,0.18,0.15,0.12,0.095,0.065,0}. Indeed, these coefficients all correspond to values in [0,0.5[. When used as coefficients for the test polynomial *testv*, it allows the algorithm to be run only on the positive half of the torus. Thanks to this, the final decryption returns a value in $\{\frac{0}{32},\frac{1}{32},\cdots,\frac{15}{32}\} \in [0,0.5[$. Then, to cover the whole image of our StepSigmoid, we (homomorphically) multiply this result by 2, which gives us a value in $\{\frac{0}{16},\frac{1}{16},\cdots,\frac{15}{16}\}$ this time corresponding to values in [0,1]. Nevertheless, the returned value does not necessarily correspond to the desired result (as TFHE will only return values in $\mathcal{M}$). Still, it is close enough (standard deviation of 0.01556) to preserve the accuracy of the network. Indeed, by replacing the desired values with the returned ones, the accuracy of the network remains identical (93.4% of accurate predictions). Additionally, we have to choose the interpretation of the input values. For simplicity's sake, we decided to take the antecedents of the values given by StepSigmoid for creating a bijection between these values and the values in $\{\frac{0}{32},\cdots,\frac{15}{32}\}$. The resulting implementation is detailed in Table 3.

4.4 Performance Results

When attempting to run an RNN (or any other type of network) in the homomorphic domain, two metrics are important: the execution time and the consequences of the required approximations on the final accuracy of the network. In our case, with respect to performance, we obtained an average execution time of 0.28 secs for a single evaluation of our StepSigmoid via the tree-based method (Sect. 4.1) and of only 0.15 secs with the MVB (Sect. 4.2). As we only change the values of the test polynomial *testv* to evaluate our StepTanh, the execution times are identical for both functions.

With respect to the accuracy, by testing our network in the plaintext domain by replacing the return values of the StepSigmoid and StepTanh with the values obtained via our homomorphic execution, the network maintains its accuracy of 93.4%. This means that the approximations chosen for the Sigmoid and Tanh are good-enough and do not significantly impact the overall accuracy of the network.

This work is a first step. However, an issue remains. Considering our non-standard interpretation of the values in $\{0,\frac{1}{16},\frac{2}{16},\cdots,\frac{15}{16}\}$ in the activation function outputs means that the subsequent addition and multiplication operations cannot be performed directly. Thus, several perspectives have to be investigated for the complete execution of the network. We can either switch back to the standard interpretation at carefully chosen points during the network execution (which may cost an additional bootstrapping per conversion) or propose new LUT-based operators for performing additions and multiplications directly in the non-standard interpretation (or a combination of

both). Both approaches will incur additional bootstrappings, which have not been considered in our coarse-grained estimations.

Value in $\mathcal{M} \times 32$	Clear interpretation	Desired result	Achieved result
0	0.12	0.53	0.5625
1	0.24	0.56	0.5625
2	0.58	0.64	0.625
3	0.85	0.70	0.6875
4	1.15	0.76	0.75
5	1.45	0.81	0.75
6	1.90	0.87	0.875
7	7.0	1.0	1.0
8	-0.12	0.47	0.5
9	-0.36	0.41	0.4375
10	-0.58	0.36	0.375
11	-0.85	0.30	0.3125
12	-1.15	0.24	0.25
13	-1.45	0.19	0.1875
14	-1.90	0.13	0.125
15	-7.0	0.0	0.0

Table 3: Evaluation of the homomorphic StepSigmoid

5 Conclusion and Perspectives

First, we have established that, unlike CNNs, LSTMs do not support rough approximations of their activation functions. The pair (Sign, Heaviside) is thus to be banished from LSTM implementations over FHE, at least without new LSTM cell designs. Moreover, our results illustrate that the discretization of the inputs and the internal coefficients of LSTMs (weight matrices and bias vectors) does not raise any issue with respect to network precision. Second, we propose approximated flavors of the activation functions of LSTMs. As shown by our experimental results applied to the Sigmoid, the accuracy of the network remains unchanged. Moreover, the proposed approximations are relatively efficient when evaluated over FHE, as it, for example, only takes a few hundredths of a second for one StepSigmoid evaluation. Finally, these unitary timings allow us to estimate the (sequential) time needed for the complete execution of our reference 128 LSTM units network using our homomorphic activation functions. The results can be found in Table 4. The given times are coarsely estimated from the execution time of a single evaluation of the activation functions. If we only count activation functions evaluation times, we obtain for the 128 LSTM units a total FHE execution time of 3 mins using the tree-based method or 1.5 mins with the MVB optimization. This gives an order of magnitude of the time required for an FHE evaluation of the complete network.

Our results are promising and open the door to an end-to-end TFHE evaluation of LSTMs with practical latencies.

	number of bootstrapping		execution time	
	tree-based method	MVB	tree-based method	MVB
single StepSigmoidexecution	5	2	0.28s	0.15s
complete LSTM unit execution	$5\times5=25$	$2\times5=10$	1.4s	0.75s
complete network execution	$128\times25=3200$	$128\times10=1280$	179s = 2min59s	96s = 1min36s

Table 4: Execution time results (number of bootstrappings are provided for informational purposes, as bootstrapping is the most costly operation in TFHE).

References

1. Aharoni, E., Adir, A., Baruch, M., Ezov, G., Farkash, A., Greenberg, L., Masalha, R., Murik, D., Soceanu, O.: Tile tensors: A versatile data structure with descriptive shapes for homomorphic encryption. CoRR **abs/2011.01805** (2020), https://arxiv.org/abs/2011.01805
2. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast Homomorphic Evaluation of Deep Discretized Neural Networks. Technical Report Report 2017/1114, IACR Cryptology ePrint Archive (Nov 2017), https://hal.science/hal-01665330
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 868–886. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. p. 309–325. ITCS '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2090236.2090262
5. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. Cryptology ePrint Archive, Paper 2018/622 (2018), https://eprint.iacr.org/2018/622
6. Chabanne, H., Lescuyer, R., Milgram, J., Morel, C., Prouff, E.: Recognition Over Encrypted Faces: 4th International Conference, MSPN 2018, Paris, France (2019)
7. 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)
8. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers (2017)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library (August 2016), https://tfhe.github.io/tfhe/
10. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. Cryptology ePrint Archive, Paper 2021/091 (2021). https://doi.org/10.1007/978-3-030-78086-91, https://eprint.iacr.org/2021/091
11. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A. (eds.) Cyber Security Cryptography and Machine Learning. pp. 1–19. Springer International Publishing, Cham (2021)
12. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. Cryptology ePrint Archive, Report 2021/729 (2021), https://ia.cr/2021/729
13. Clet, P.E., Zuber, M., Boudguiga, A., Sirdey, R., Gouy-Pailler, C.: Putting up the swiss army knife of homomorphic calculations by means of tfhe functional bootstrapping. Cryptology ePrint Archive, Paper 2022/149 (2022), https://eprint.iacr.org/2022/149

14. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Tech. Rep. MSR-TR-2016-3 (February 2016), https://www.microsoft.com/en-us/research/publication/cryptonets-applying-neural-networks-to-encrypted-data-with-high-throughput-and-accuracy/
15. Dua, M., Yadav, R., Mamgai, D., Brodiya, S.: An improved RNN-LSTM based novel approach for sheet music generation. Procedia Computer Science **171**, 465–474 (01 2020). https://doi.org/10.1016/j.procs.2020.04.049
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://ia.cr/2012/144
17. Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in tfhe. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(2), 229–253 (Feb 2021). https://doi.org/10.46586/tches.v2021.i2.229-253
18. Hochreiter, and Jurgen, S.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
19. Izabachène, M., Sirdey, R., Zuber, M.: Practical fully homomorphic encryption for fully masked neural networks. In: Mu, Y., Deng, R.H., Huang, X. (eds.) Cryptology and Network Security. pp. 24–36. Springer International Publishing, Cham (2019)
20. Jang, J., Lee, Y., Kim, A., Na, B., Yhee, D., Lee, B., Cheon, J.H., Yoon, S.: Privacy-preserving deep sequential model with matrix homomorphic encryption. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. p. 377–391. ASIA CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3488932.3523253
21. 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)
22. Kluczniak, K., Schild, L.: FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/1135 (2021), https://ia.cr/2021/1135
23. Lev, G., Sadeh, G., Klein, B., Wolf, L.: Rnn fisher vectors for action recognition and image annotation (12 2015)
24. Madi, A., Sirdey, R., Stan, O.: Computing neural networks with homomorphic encryption and verifiable computing. In: ACNS Workshops (2020)
25. OPenAI: Chatgpt: Optimizing language models for dialogue (2022), https://openai.com/blog/chatgpt/
26. Paul, J., Annamalai, M.S.M.S., Ming, W., Badawi, A.A., Veeravalli, B., Aung, K.M.M.: Privacy-preserving collective learning with homomorphic encryption. IEEE Access **9**, 132084–132096 (2021). https://doi.org/10.1109/ACCESS.2021.3114581
27. Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., Chen, M.: Hierarchical text-conditional image generation with clip latents Http://arxiv.org/abs/2204.06125
28. Syed, S.A., Rashid, M., Hussain, S., Zahid, H.: Comparative analysis of cnn and RNN for voice pathology detection. BioMed Research International **2021** (2021)
29. Woodbridge, J., Anderson, H.S., Ahuja, A., Grant, D.: Predicting domain genreation algorithms with long short-term memory networks (2016)
30. Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: Tota: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Report 2021/1347 (2021), https://ia.cr/2021/1347