Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping

Anonymous Submission

Abstract. In this work, we first propose a new full domain functional bootstrapping method with TFHE for evaluating any function of domain and codomain the real torus $\mathbb{T}$ by using a small number of bootstrappings. This result improves some aspects of previous approaches: like them, we allow for evaluating any functions, but with better precision. In addition, we develop efficient multiplication and addition over ciphertexts building on the digit-decomposition approach of [GBA21]. As a practical application, our results lead to an efficient implementation of ReLU, one of the most used activation functions in deep learning. The paper is concluded by extensive experimental results comparing each building block as well as their practical relevance and trade-offs.

Keywords: FHE · TFHE · functional bootstrapping

1 Introduction

Machine learning application to the analysis of private data, such as health or genomic data, has encouraged the use of homomorphic encryption for private inference or prediction with classification or regression algorithms where the ML models and/or their inputs are encrypted homomorphically [Xie+14; Cha+17; Cha+19; Bou+18; ZCS20b; ISZ19; ZS21]. Even training machine learning models with privacy guarantees on the training data has been investigated in the centralized [JA18; CKP19; Nan+19; Lou+20] and collaborative [Séb+21; Mad+21] settings. In practice, machine learning algorithms and especially neural networks require the computation of non-linear activation functions such as the sign, ReLU or sigmoid functions. Computing non-linear functions homomorphically remains challenging. For levelled homomorphic schemes such as BFV [Bra12; FV12] or CKKS [Che+17], non-linear functions have to be approximated by polynomials. However, the precision of this approximation differs with respect to the considered plaintext space (i.e., input range), approximation polynomial degree and its coefficients size, and has a direct impact on the multiplicative depth and parameters of the cryptosystem. The more precise is the approximation, the larger are the cryptosystem parameters and the slower is the computation. On the other hand, homomorphic encryption schemes having an efficient bootstrapping, such as TFHE [Chi+16; Chi+19] or FHEW [DM15], can be tweaked to encode functions via look-up table evaluations within their bootstrapping procedure. Hence, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), the bootstrapping becomes *functional* [BST19] or *programmable* [CJP21] by allowing the evaluation of arbitrary functions as a bonus. These capabilities results in promising new approaches for improving the overall performances of homomorphic calculations, making the FHE "API" better suited to the evaluation of mathematical operators which are difficult to express as low complexity arithmetic circuits. It is also important to note that FHE cryptosystems can be hybridized, for example BFV ciphertexts can be efficiently (and homomorphically) turned into TFHE ones [Bou+20; ZCS20a]. As such, the building blocks discussed in this paper are of relevance also in the setting where the desired encrypted-domain calculation can be split into a preprocessing step more efficiently done using BFV (e.g. several dot product or distance computations) followed by a nonlinear postprocessing step (such as an activation function or an argmin) which can then be more conveniently performed by exploiting TFHE functional bootstrapping. In this work, we thus systematize and further investigate the capabilities of TFHE functional bootstrapping.

Contributions – In this paper, we review, unify and extend the capabilities of TFHE functional bootstrapping. We strive to present the main existing methods as well as new variants. We compare their relative accuracy and performance as well as discuss their main pros and cons. Indeed, on top of the extensions that we present, we aim for this paper to be a complete reference for anyone looking to get a view of the state of functional bootstrapping. As such, several methods for LUTs evaluation using functional bootstrapping are presented: the usual method using one bit of padding (described clearly in [CJP21]), two methods coming from recent papers that work without padding [KS21; Yan+21], one novel approach also working without padding, and a method using digit decomposition of the inputs in order to get an arbitrary large plaintext space (presented initially by Bourse et al., [BST19] and generalized later by Guimarães et al. [GBA21]). The first method encodes the plaintext space in $[0, \frac{1}{2}[$, i.e., the segment of the real torus $\mathbb{T}$ corresponding to the positive numbers. Meanwhile, the other methods use the full torus for encoding the plaintext space and propose various solutions to cope with the negacyclicity of TFHE bootstrapping when used for evaluating LUTs. A novel way we present to achieve this is to use several bootstrappings one after the other to cancel the negacyclicity of a single bootstrapping. Finally, the decomposition method allows working with larger plaintext spaces. Its main idea is to decompose each plaintext into small digits which allows keeping TFHE parameters small enough to lead to performance improvements. We generalize the chaining method of [GBA21] in order to compute *any* function with *any* chosen precision.

Related works – In 2016, the TFHE paper made a breakthrough by proposing an efficient bootstrapping for homomorphic gate computation. Then, Bourse et al., [Bou+18] and Izabachene et al., [ISZ19] used the same bootstrapping algorithm for extracting the (encrypted) sign of an encrypted input. Boura et al., [Bou+19] showed later that TFHE bootstrapping could be extended to support a wider class of functionalities. Indeed, TFHE bootstrapping naturally allows to encode function evaluation via their representation as look-up tables (LUTs). Recently, different approaches have been investigated for functional bootstrapping improvement. In particular, Kluczniak and Schild [KS21] and Yang et al., [Yan+21] proposed two methods that take into consideration the negacyclicity of the cyclotomic polynomial used within the bootstrapping, for encoding look-up tables over the full real torus $\mathbb{T}$. Meanwhile, Guimarães et al., [GBA21] extended the ideas in Bourse et al., [BST19] to support the evaluation of certain activation functions such as the $\mathsf{sigmoid}$. One last method, presented in Chillotti et al., [Chi+21] achieves a functional bootstrapping over the full torus using a BFV type multiplication.

Paper organization – The remainder of this paper is organized as follows. Section 2 reviews TFHE building blocks. Section 3 describes the functional bootstrapping idea coming from the TFHE gate bootstrapping. Sections 4 and 5 detail several methods, including ours, for the intricate Look-Up Tables (LUTs) encoding via the functional bootstrapping. Indeed, section 4 describes methods for LUTs evaluation when having a unique ciphertext as input. Meanwhile, section 5 considers the case where LUTs are evaluated over several ciphertexts encrypting separately the digits of a large plaintext. Finally, section 6 gives unitary results comparing these methods for LUTs evaluation over encrypted data.

2 TFHE

2.1 Notations

In the upcoming sections, we denote vectors by bold letters and so, each vector $\boldsymbol{x}$ of n elements is described as: $\boldsymbol{x}=(x_1,...,x_n)$. $\langle \boldsymbol{x},\boldsymbol{y}\rangle$ is the dot product between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. We denote matrices by capital letters, and the set of matrices with m rows and n columns with entries sampled in $\mathbb{K}$ by $\mathcal{M}_{m,n}(\mathbb{K})$. $x \xleftarrow{\$} \mathbb{K}$ denotes sampling x uniformly from $\mathbb{K}$, while $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{K}$ refers to sampling x from $\mathbb{K}$ following a Gaussian distribution of mean μ and variance σ^2.

We will use the same notations for parameters as in the TFHE article [Chi+19].

We will refer to the real torus by $\mathbb{T}=\mathbb{R}/\mathbb{Z}$. $\mathbb{T}$ is the additive group of real numbers modulo 1 ($\mathbb{R}$ $mod[1]$) and it is a $\mathbb{Z}$-module. That is, multiplication by scalars from $\mathbb{Z}$ is well-defined over $\mathbb{T}$. $\mathbb{T}_N[X]$ denotes the $\mathbb{Z}$-module $\mathbb{R}[X]/(X^N+1)$ $mod[1]$ of torus polynomials, where N is a power of 2. $\mathcal{R}$ is the ring $\mathbb{Z}[X]/(X^N+1)$ and its subring of polynomials with binary coefficients is $\mathbb{B}_N[X]=\mathbb{B}[X]/(X^N+1)$ ($\mathbb{B}=\{0,1\}$). Finally, $[x]$ will denote the encryption of x over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$. x is sampled from the plaintext set $\mathcal{M}$ of cardinality $|\mathcal{M}|$.

Given a function $f:\mathbb{T}\to\mathbb{T}$, we define $\mathrm{LUT}_N(f)$ to be Look-Up Table defined by the set of N pairs $\left(i,f\left(\frac{i}{N}\right)\right)$. We may write $\mathrm{LUT}(f)$ when the value N is implied. Given a function $f:\mathbb{T}\to\mathbb{T}$, we define a polynomial $P_{f,N}\in\mathbb{T}_N[X]$ of degree N by writing $P_{f,N}=\sum_{i=0}^{N-1} f\left(\frac{i}{2N}\right)\cdot X^i$. For simplicity sake, we may write P_f instead of $P_{f,N}$ when the value N is implied.

2.2 TFHE Structures

The TFHE encryption scheme was proposed in 2016 [Chi+16]. It improves the FHEW cryptosystem [DM15] and introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. It was updated later in [Chi+17] and both works were recently unified in [Chi+19]. The TFHE scheme is implemented as the TFHE library [Chi+]. TFHE relies on three structures to encrypt plaintexts defined over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$:

- **TLWE Sample:** $(\boldsymbol{a},b)$ is a valid TLWE sample if $\boldsymbol{a}\xleftarrow{\$}\mathbb{T}^n$ and $b\in\mathbb{T}$ verifies $b=\langle\boldsymbol{a},\boldsymbol{s}\rangle+e$, where $\boldsymbol{s}\xleftarrow{\$}\mathbb{B}^n$ is the secret key, and $e\xleftarrow{\mathcal{N}(0,\sigma^2)}\mathbb{T}$. Then, $(\boldsymbol{a},b)$ is a fresh encryption of 0.
- **TRLWE Sample:** a pair $(\boldsymbol{a},b)\in\mathbb{T}_N[X]^k\times\mathbb{T}_N[X]$ is a valid TRLWE sample if $\boldsymbol{a}\xleftarrow{\$}\mathbb{T}_N[X]^k$, and $b=\langle\boldsymbol{a},\boldsymbol{s}\rangle+e$, where $\boldsymbol{s}\xleftarrow{\$}\mathbb{B}_N[X]^k$ is a TRLWE secret key and $e\xleftarrow{\mathcal{N}(0,\sigma^2)}\mathbb{T}_N[X]$ is a noise polynomial. In this case, $(\boldsymbol{a},b)$ is a fresh encryption of 0.

  The TRLWE decision problem consists of distinguishing TRLWE samples from random samples in $\mathbb{T}_N[X]^k\times\mathbb{T}_N[X]$. Meanwhile, the TRLWE search problem consists in finding the private polynomial $\boldsymbol{s}$ given arbitrarily many TRLWE samples. When $N=1$ and k is large, the TRLWE decision and search problems become the TLWE decision and search problems, respectively.

  Let $\mathcal{M}\subset\mathbb{T}_N[X]$ (or $\mathcal{M}\subset\mathbb{T}$) be the discrete message space[1]. To encrypt a message $m\in\mathcal{M}\subset\mathbb{T}_N[X]$, we add $(\boldsymbol{0},m)\in\mathbb{T}_N[X]^k\times\mathbb{T}_N[X]$ to a TRLWE sample encrypting 0 (or to a TLWE sample of 0 if $\mathcal{M}\subset\mathbb{T}$). In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a T(R)LWE ciphertext noted $\boldsymbol{c}\in$ T(R)LWE$_{\boldsymbol{s}}(m)$.

  To decrypt a sample $\boldsymbol{c}\in$ T(R)LWE$_{\boldsymbol{s}}(m)$, we compute its *phase* $\phi(\boldsymbol{c})=b-\langle\boldsymbol{a},\boldsymbol{s}\rangle=m+e$. Then, we round to it to the nearest element of $\mathcal{M}$. Therefore, if the error e was chosen to

[1] In practice, we discretize the Torus with respect to our plaintext modulus. For example, if we want to encrypt $m\in\mathbb{Z}_4=\{0,1,2,3\}$, we encode it in $\mathbb{T}$ as one of the following value $\{0,0.25,0.5,0.75\}$.

be small enough (yet high enough to ensure security), the decryption will be accurate.

- **TRGSW Sample:** is a vector of l TRLWE samples encrypting 0. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample of 0, where H is a gadget matrix[2]. Chilotti et al., [Chi+19] defines an external product between a TRGSW sample A encrypting $m_a \in \mathcal{R}$ and a TRLWE sample $\boldsymbol{b}$ encrypting $m_b \in \mathbb{T}_N[X]$. This external product consists in multiplying A by the approximate decomposition of $\boldsymbol{b}$ with respect to H (Definition 3.12 in [Chi+19]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE sample $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \mathrm{TRGSW}_{\boldsymbol{s}}(b), b \in \{0,1\}$, and the inputs are $\boldsymbol{c}_0 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_1)$.

2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as a TRLWE ciphertext by an encrypted position. It takes as inputs: a TRLWE ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{k}}(m)$, a vector $(a_1,...,a_p,a_{p+1}=b)$ where $\forall i$, $a_i \in \mathbb{Z}_{2N}$, and p TRGSW ciphertexts encrypting $(s_1,...,s_p)$ where $\forall i$, $s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \mathrm{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a},\boldsymbol{s}\rangle - b} \cdot m)$. In this paper, we will refer to this algorithm by BlindRotate.

- **TLWE Sample Extract:** takes as inputs a ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{k}}(m)$ and a position $p \in [\![0,N[\![$, and returns a TLWE ciphertext $\boldsymbol{c'} \in \mathrm{TLWE}_{\boldsymbol{k}}(m_p)$ where m_p is the p^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm by SampleExtract.

- **Public Functional Keyswitching:** transforms a set of p ciphertexts $\boldsymbol{c}_i \in \mathrm{TLWE}_{\boldsymbol{k}}(m_i)$ into a ciphertext $\boldsymbol{c'} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(f(m_1,...,m_p))$, where $f()$ is a public linear morphism from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. Note that functional keyswitching serves at changing encryption keys and parameters. In this paper, we will refer to this algorithm by KeySwitch.

TFHE comes with two bootstrapping algorithms. The first one is the gate bootstrapping. It aims at reducing the noise level of a TLWE sample that encrypts the result of a boolean gate evaluation on two ciphertexts, each of them encrypting a binary input. The binary nature of inputs/outputs of this algorithm is not due to inherent limitations of the TFHE scheme but rather to the fact that the authors of the paper were building a bitwise set of operators for which this bootstrapping operation was perfectly fitted.

TFHE gate bootstrapping steps are summarized in Algorithm 1. The step 1 consists in selecting a value $\hat{m} \in \mathbb{T}$ which will serve later for setting the coefficients of the test polynomial *testv* (in step 3). The step 2 rescales the components of the input ciphertext $\boldsymbol{c}$ as elements of $\mathbb{Z}_{2N}$. The step 3 defines the test polynomial *testv*. Note that for all $p \in [\![0,2N[\![$, the constant term of $testv \cdot X^p$ is $\hat{m}$ if $p \in]\!]\frac{N}{2},\frac{3N}{2}]\!]$ and $-\hat{m}$ otherwise. The step 4 returns an accumulator $ACC \in \mathrm{TRLWE}_{\boldsymbol{s'}}(testv \cdot X^{\langle \bar{\boldsymbol{a}},\boldsymbol{s}\rangle - \bar{b}})$. Indeed, the constant term of ACC is $-\hat{m}$ if $\boldsymbol{c}$ encrypts 0, or $\hat{m}$ if $\boldsymbol{c}$ encrypts 1. Then, step 5 creates a new ciphertext $\bar{\boldsymbol{c}}$ by extracting the constant term of ACC and adding to it $(\boldsymbol{0},\hat{m})$. That is, $\bar{\boldsymbol{c}}$ either encrypts 0 if $\boldsymbol{c}$ encrypts 0, or m if $\boldsymbol{c}$ encrypts 1 (By choosing $m=\frac{1}{2}$, we get a fresh encryption of 1).

TFHE specifies a second type of bootstrapping called *circuit bootstrapping*. It converts TLWE samples into TRGSW samples, and serves mainly for TFHE use in a levelled manner.

[2] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [Chi+19] for more information about the gadget matrix H.

Algorithm 1 TFHE gate bootstrapping [Chi+19]

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

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \text{TLWE}_{\boldsymbol{s}}(x.m)$

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

2.4 Error Variance and Rate

In this section, we remind results from [Chi+19] regarding the error's variance for the BlindRotate and KeySwitch functions from Algorithm 1. These results will serve later to bound the errors' variance and rate for the discussed functional bootstrapping algorithms.

Proposition 1. *Let $\overline{\boldsymbol{c}}$ be the output of Algorithm 1 when taking as input a TLWE ciphertext $\boldsymbol{c}$ (without considering the* KeySwitch *i.e., without line 6 of Algorithm 1). Then, the variance of the noise of $\overline{\boldsymbol{c}}$, $Var(Err(\overline{\boldsymbol{c}}))$, is bounded by:*

$$Var(Err(\overline{\boldsymbol{c}})) \leq n((k+1)\ell N (\frac{B_g}{2})^2 \vartheta_{BK} + \frac{(1+kN)}{4 \cdot B_g^{2l}})$$

where ϑ_{BK} is the variance of the bootstrapping key, and B_g and l are the decomposition parameters of the gadget matrix H. B_g is the decomposition base and l serves to compute the decomposition precision $\epsilon = \frac{1}{2 \cdot B_g^l}$.

Proof. This result is a direct consequence of the noise analysis for BlindRotate. Please refer to [Chi+19] for the complete proof. □

In the following, we will refer to the error bound by $\mathcal{E}_{BS}$:

$$\mathcal{E}_{BS} = n((k+1)\ell N (\frac{B_g}{2})^2 \vartheta_{BK} + \frac{(1+kN)}{4 \cdot B_g^{2l}})$$

Proposition 2. *Given $\boldsymbol{c}$ a TLWE ciphertext encrypting a message m from the discrete message space $\mathcal{M}$, the probability of error for the bootstrapping algorithm, when taking as input $\boldsymbol{c}$ verifies:*

$$P(Err(\boldsymbol{c})) = 1 - erf(\frac{1}{2 \cdot |\mathcal{M}| \cdot \sqrt{V_c + V_r} \cdot \sqrt{2}})$$

where $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is the Gaussian error function, V_c is the variance of $Err(\boldsymbol{c})$, and $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in the bootstrapping algorithm (line 2 of Algorithm 1).

Proof. The probability of bootstrapping error is the complementary to 1 of the probability of BlindRotate success. The latter is the probability that the sum of the input ciphertext noise with the rounding error (from line 2 of Algorithm 1) is smaller than half the interval allocated to a given value on the torus. That is, we need the noise of $\frac{\lfloor 2N\boldsymbol{c} \rceil}{2N}$ to be smaller than $\frac{1}{2|\mathcal{M}|}$. Let's consider that $\frac{\lfloor 2N\boldsymbol{c} \rceil}{2N} = \boldsymbol{c} + \boldsymbol{r}$ where $\boldsymbol{r}$ is an error that comes from the rounding operation. The proposition result is obtained from the properties of the *erf* function and the fact that the variance of $\boldsymbol{c}+\boldsymbol{r}$ is equal to the sum of their separate variances. ☐

The KeySwitch operation (line 6) at the end of the bootstrapping Algorithm 1 does not change the probability of error of the algorithm. However, it does change the resulting noise. The following proposition bounds the variance of the KeySwitch noise.

Proposition 3. *Let $\overline{\mathbf{c}}$ be the output of the* KeySwitch *algorithm when it takes as input the TLWE ciphertext $\boldsymbol{c}$. Then, the variance of the noise of $\overline{\mathbf{c}}$ is:*

$$Var(Err(\overline{\mathbf{c}})) \leq R^2\, Var(Err(\boldsymbol{c})) + n(tN\vartheta_{KS} + \frac{B_{KS}^{-2t}}{12})$$

where ϑ_{KS} is the variance of the keyswitching key, R is the Lipschitz constant of the linear application computed during the keyswitching operation. It will be always equal to 1 *in our paper. B_{KS} is a decomposition base, and t sets the decomposition precision to $\epsilon_{KS} = \frac{1}{2B_{KS}^t}$.*

Proof. Please refer to [Chi+19] for a proof of this result with $B_{KS}=2$ and to [GBA21] for a generalization to any decomposition base. ☐

In the following, we set the KeySwitch error bound to $\mathrm{Var}(\mathrm{Err}(\boldsymbol{c})) + \mathcal{E}_{KS}$, where:

$$\mathcal{E}_{KS} = n(tN\vartheta_{KS} + \frac{B_{KS}^{-2t}}{12})$$

Proposition 4. *Let $\overline{\mathbf{c}}$ be the output of the bootstrapping Algorithm 1 when it takes as input the TLWE ciphertext $\boldsymbol{c}$. Then, the variance of the error of $\overline{\mathbf{c}}$ verifies:*

$$Var(Err(\overline{\mathbf{c}})) \leq \mathcal{E}_{BS} + \mathcal{E}_{KS}$$

Proof. The result comes directly from the combination of propositions 1 and 3. ☐

3 TFHE Functional Bootstrapping

3.1 Encoding and Decoding

Our goal is to build an homomorphic LUT of any function $f:\mathcal{I}\to\mathcal{O}$ with varying precision and with input and output spaces $\mathcal{I},\mathcal{O}\subset\mathbb{R}$.

Since we use TFHE as our homomorphic encryption scheme, every message from plaintext input or output space needs to be encoded in $\mathbb{T}$. Therefore, in order to build our function f, we need to create a torus-to-torus function $f_{\mathbb{T}}$ and appropriate encoding and decoding functions ι and ω.

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{f=\omega\circ f_{\mathbb{T}}\circ\iota} & \mathcal{O} \\ \iota\downarrow & & \uparrow\omega \\ \mathbb{T} & \underset{f_{\mathbb{T}}}{\longrightarrow} & \mathbb{T} \end{array}$$

In most cases, ι and ω are rescaling functions: a multiplication or a division by a single fixed value. In the following, we show several ways to build any Look-Up Table (LUT) evaluating function $f_{\mathbb{T}}$.

3.2 Functional Bootstrapping Idea

The original bootstrapping algorithm from [Chi+16] had already all the tools to implement a LUT of any negacyclic function[3] In particular, TFHE is well-suited for $\frac{1}{2}$-antiperiodic function, as the plaintext space for TFHE is $\mathbb{T}$, where $[0,\frac{1}{2}[$ corresponds to positive values and $[\frac{1}{2},1[$ to negative ones, and the bootstrapping step 2 of the Algorithm 1 encodes elements from $\mathbb{T}$ into powers of X modulus (X^N+1). Note that $X^{\alpha+N} \equiv -X^{\alpha} mod[X^N+1]$ and allows encoding negacylic functions as explained in the upcoming sections.

Boura et al., [Bou+19] were the first to use the term *functional bootstrapping* for TFHE. They describe how TFHE bootstrapping computes a sign function. In addition, they state that bootstrapping can be used to build a Rectified Linear Unit (ReLU). However, they do not delve into the details of how to implement the ReLU in practice[4].

Algorithm 2 describes a sign computation with the TFHE bootstrapping. It returns μ if m is positive (i.e., $m \in [0,\frac{1}{2}[$), and $-\mu$ if m is negative.

Algorithm 2 Sign extraction with bootstrapping

Input: a constant $\mu \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s}\rightarrow\boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$

Output: a TLWE sample $\bar{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(\mu.sign(m))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
2: Let $testv := (1+X+\cdots+X^{N-1})\cdot\mu \in \mathbb{T}_N[X]$
3: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0},testv),(\bar{a}_1,\ldots,\bar{a}_n,\bar{b}),(BK_1,\ldots,BK_n))$
4: $\bar{\boldsymbol{c}} = \mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\mathsf{s'}\rightarrow\mathsf{s}}(\bar{\boldsymbol{c}})$

When we look at the building blocks of Algorithm 2, we notice that there is some leeway to build more complex functions just by changing the coefficients of the test polynomial $testv$.

Let $t = \sum_{i=0}^{N-1} t_i \cdot X^i$ where $t_i \in \mathbb{T}$ and $g_t(x)$ the function:

$$g_t : \begin{array}{ccc} [\![-N,N-1]\!] & \rightarrow & \mathbb{T} \\ i & \mapsto & \begin{cases} t_i & \text{if } i \in [\![0,N[\![\\ -\,t_{i+N} & \text{if } i \in [\![-N,0[\![\end{cases} \end{array}$$

Proposition 5. *If we bootstrap a TLWE ciphertext $[x] = (\boldsymbol{a},b)$ with the test polynomial $testv = t$, the output of the bootstrapping is $[g_t(\phi(\bar{\boldsymbol{a}},\bar{b}))]$, where $(\bar{\boldsymbol{a}},\bar{b})$ is the rescaled version of $(\boldsymbol{a},b)$ in $\mathbb{Z}_{2N}$ (line 1 of Algorithm 2).*

Proof. First, we remind that for any positive integer i s.t. $0 \le i < N$, we have:

$$testv.X^{-i} = t_i + \cdots - t_0 X^{N-i} - \cdots - t_{i-1} X^{N-1} \quad \mathrm{mod}[X^N+1] \tag{1}$$

Then, we notice that $\mathsf{BlindRotate}$ (line 3 of Algorithm 2) computes $testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})}$. Therefore, we obtain the following results using equation (1):

- if $\phi(\bar{\boldsymbol{a}},\bar{b}) \in [\![0,N[\![$, the constant term of $testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})}$ is $t_{\phi(\bar{\boldsymbol{a}},\bar{b})}$.

[3] Negacyclic functions are antiperiodic functions over $\mathbb{T}$ with period $\frac{1}{2}$, i.e., verifying $f(x) = -f(x+\frac{1}{2})$.

[4] The article does only mention that the function $2\times\mathsf{ReLU}$ can be built from an absolute value function but does not explain how to divide by two to get the ReLU result.

- if $\phi(\bar{\boldsymbol{a}},\bar{b}) \in [\![-N,0[\![$, we have:

  $testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})} = -testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})-N} mod[X^N+1]$

  with $(\phi(\bar{\boldsymbol{a}},\bar{b})+N) \in [\![0,N[\![$. So, the constant term of $testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})}$ is $-t_{\phi(\bar{\boldsymbol{a}},\bar{b})+N}$.

All that remains for the bootstrapping algorithm is extracting the previous constant term (in line 4) and keyswitching (in line 5) to get the TLWE sample $[g_t(\phi(\bar{\boldsymbol{a}},\bar{b}))]$. $\square$

We can use the previous proposition to build a discretized function evaluation as follows. Let $h:[0,\frac{1}{2}[\to \mathbb{T}$ be any function, and g_h the well-defined function:

$$g_h : \begin{array}{ccc} [\![-N,N-1]\!] & \to & \mathbb{T} \\ x & \mapsto & \begin{cases} h(\frac{x}{2N}) & \text{if } x \in [\![0,N[\![\\ -\,h(\frac{x+N}{2N}) & \text{if } x \in [\![-N,0[\![\end{cases} \end{array} \tag{2}$$

Let's call P_h the polynomial of degree N defined by: $P_h = \sum_{i=0}^{N-1} h\left(\frac{i}{2N}\right) \cdot X^i$. Now, if we apply the bootstrapping Algorithm 2 to a TLWE ciphertext $[x] = (\boldsymbol{a},b)$ with $testv = P_h$, it outputs $[g_h(\phi(\bar{\boldsymbol{a}},\bar{b}))]$ (by applying Proposition 5). That is, Algorithm 2 allows encoding a discretized negacyclic version of h. In that way, it allows encoding a discretized version of any negacyclic function.

3.3 Private Functional Bootstrapping

The functional bootstrapping algorithm can be adapted to compute an encrypted negacyclic function. Indeed, given a function $f:\mathbb{T}\to\mathbb{T}$, we create $[P_f]$, a TRLWE ciphertext whose i^{th} coefficient is a TLWE ciphertext encrypting $f(\frac{i}{2N})$. Such a ciphertext can be created using the TFHE public functional key-switching operation (see Algorithm 2 of [Chi+19]) from N TLWE ciphertexts $\left[f(\frac{i}{2N})\right]$.

Let $\boldsymbol{c} = (\boldsymbol{a},b)$ be a ciphertext encrypting the message μ. Then, the Algorithm 3 outputs an encryption of $f(\frac{\phi(\bar{\boldsymbol{a}},\bar{b})}{2N})$.

Algorithm 3 Encrypted LUT

Input: a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \text{TLWE}_{\boldsymbol{s}}(\mu)$ with $\mu \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \text{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$, an encryption $[P_f]$ of the polynomial P_f

Output: a TLWE sample $\bar{\boldsymbol{c}} \in \text{TLWE}_{\boldsymbol{s}}(f(\frac{\phi(\bar{\boldsymbol{a}},\bar{b})}{2N}))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
2: Let $testv := [P_f]$
3: $ACC \leftarrow \mathsf{BlindRotate}(testv,(\bar{a}_1,\ldots,\bar{a}_n,\bar{b}),(BK_1,\ldots,BK_n))$
4: $\bar{\boldsymbol{c}} = \mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\mathbf{s'}\to\mathbf{s}}(\bar{\boldsymbol{c}})$

Proposition 6. *Let $\bar{\boldsymbol{c}}$ be the output of the private functional bootstrapping algorithm when given as input $\boldsymbol{c}$. Then, the variance of the noise of $\bar{\boldsymbol{c}}$ verifies:*

$$Var(Err(\bar{\boldsymbol{c}})) \leq Var(Err([P_f])) + \mathcal{E}_{BS} + \mathcal{E}_{KS}$$

Proof. This result corresponds to the combination of the variance of the errors of BlindRotate and KeySwitch. The term Var(Err($[P_f]$)) comes from the BlindRotate error [Chi+19]. $\square$

Note that the term $\mathrm{Var}(\mathrm{Err}([P_f]))$ was equal to 0 for Algorithms 1 and 2 as we were using a noiseless and trivial TRLWE sample $(\mathbf{0}, testv)$ as input for the BlindRotate.

Proposition 7. *Let* $\mathbf{c}$ *be a TLWE ciphertext, and suppose that we apply a negacyclic LUT which differentiates* $|\mathcal{M}|$ *possible input values, the probability of error of the private functional bootstrapping algorithm with* $\mathbf{c}$ *as input verifies:*

$$P(Err(\mathbf{c})) = 1 - erf\left(\frac{1}{2 \cdot |\mathcal{M}| \cdot \sqrt{V_c + V_r} \cdot \sqrt{2}}\right)$$

where $V_r = \frac{n+1}{48N^2}$ *is the variance of the error induced by the rounding operation in line 1 of Algorithm 3.*

Proof. The proof is the same as for Proposition 2. □

3.4 Multi-Value Functional Bootstrapping

Carpov et al., [CIM19] introduced a nice method for evaluating k different LUTs using one bootstrapping. Indeed, they factor the test polynomial P_{f_i} associated to 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 fact, they notice that:

$$(1 + X + \cdots + X^{N-1}) \cdot (1 - X) = 2 \mod [X^N + 1] \tag{3}$$

Let's write P_{f_i} as: $P_{f_i} = \sum_{j=0}^{N-1} \alpha_{i,j} X^j$ with $\alpha_{i,j} \in \mathbb{Z}$. We obtain using equation (3):

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

where:

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

Thanks to this factorization, we are able to compute many LUTs with one bootstrapping. Indeed, we just have to set the initial test polynomial to $testv = v_0$ during the bootstrapping. Then, after the BlindRotate, we multiply the obtained ACC by each v_i corresponding to $\mathrm{LUT}(f_i)$ to obtain ACC_i (for more details about multi-value bootstrapping and error analysis, refer to Algorithm 7 in Appendix Section A).

4 Look-Up-Tables over a Single Ciphertext

In Section 3.2, we demonstrated that functional bootstrapping allows for the computation of $\mathrm{LUT}(h)$ for any negacyclic function h. In this section, we describe 4 different ways to build homomorphic LUTs using *any* function (i.e., not necessarily negacyclic ones). We present 3 solutions from the state of the art [CJP21; KS21; Yan+21] in Sections 4.1, 4.2 and 4.3, and one that is novel to our work in Section 4.4.

As in Section 3.1, we call $f_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$ the function used to build our homomorphic LUT, and $f : \mathcal{I} \to \mathcal{O}$ its corresponding function over the actual input and output spaces.

4.1 Partial Domain Functional Bootstrapping

This method avoids the negacyclic restriction of functional bootstrapping by encrypting values from $[0, \frac{1}{2}[$ (i.e., half of the torus). Let's set the test polynomial to be P_h, the output of the bootstrapping operation is given by Equation 2:

$$g_h : \begin{array}{ccc} [\![-N, N-1]\!] & \to & \mathbb{T} \\ x & \mapsto & \begin{cases} h(\frac{x}{2N}) & \text{if } x \in [\![0, N[\![\\ -h(\frac{x+N}{2N}) & \text{if } x \in [\![-N, 0[\![\end{cases} \end{array}$$

If we restrict g_h domain to $[\![0, N[\![$, we ensure that g_h is just a LUT based on function h (h is not necessarily negacyclic). That is, we obtain a method to evaluate a LUT in a *single* bootstrapping. However, we have to encode the plaintext space over a smaller portion of the torus $\mathbb{T}$, therefore increasing the relative noise introduced by the TFHE encryption process. The overall result will hence be less accurate.

Proposition 8. *Let $\overline{\mathbf{c}}$ be the output of the partial domain functional bootstrapping algorithm for a given input. Then, the variance of the error of $\overline{\mathbf{c}}$ verifies:*

$$Var(Err(\overline{\mathbf{c}})) \leq \mathcal{E}_{BS} + \mathcal{E}_{KS}$$

Proof. This result is a direct application of Proposition 1. □

Proposition 9. *Let $\mathbf{c}$ be a TLWE ciphertext, and suppose that we differentiate $|\mathcal{M}|$ possible input values over half of the torus. The probability of error of the partial domain functional bootstrapping algorithm with input $\mathbf{c}$ verifies:*

$$P(Err(\boldsymbol{c})) = 1 - erf\left(\frac{1}{4 \cdot |\mathcal{M}| \cdot \sqrt{V_c + V_r} \cdot \sqrt{2}}\right)$$

where $V_r = \frac{n+1}{48N^2}$ is the standard deviation of the error induced by the rounding operation in the bootstrapping algorithm.

Proof. This result is a direct application of Proposition 2. □

4.2 Full Domain Functional Bootstrapping–FDFB

Kluczniak and Schild [KS21] proposed this method to evaluate encrypted LUTs of domain the whole torus $\mathbb{T}$. Let's consider a TLWE ciphertext $[m]$ encrypting the message m, and a function f of domain $\mathbb{T}$. We denote by g the function:

$$g : \begin{array}{ccc} \mathbb{T} & \to & \mathbb{T} \\ x & \mapsto & -f(x + \frac{1}{2}) \end{array}$$

We define the Heaviside function H as:

$$H : x \mapsto \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

H can be expressed using the sign function as follows: $H(x) = \frac{\text{sign}(x)+1}{2}$.

First, we compute $[H(m)]$ with only one bootstrapping (using Algorithm 2) and deduce $[(1-H)(m)] = [1] - [H(m)]$, where $[1]$ is a noiseless and trivial TLWE sample encrypting 1.

Keep in mind that 1 is represented over the torus by $\frac{1}{\mathcal{M}}$. Then, we make a keyswitch to transform the TLWE sample $[(1-H)(m)]$ into a TRLWE sample. Finally, we define:

$$\boldsymbol{c}_{\mathrm{LUT}} = (P_g - P_f)\cdot[(1-H)(m)] + (\mathbf{0}, P_f)$$

$$\boldsymbol{c}_{\mathrm{LUT}} = \begin{cases} [P_f] & \text{if } m \geq 0 \\ [P_g] & \text{if } m < 0 \end{cases}$$

Note that depending on the sign of m, $\boldsymbol{c}_{\mathrm{LUT}}$ is a TRLWE encryption of P_f or P_g, the test polynomials of f or g, respectively. Indeed, after a functional bootstrapping of $[m]$ using $\boldsymbol{c}_{\mathrm{LUT}}$ as a test polynomial, we obtain $[f(m)]$. This functional bootstrapping requires 2 BlindRotate during the bootstrapping: one to compute the Heaviside function and the other to apply the encrypted LUT. In addition, we can reduce the noise of $\boldsymbol{c}_{\mathrm{LUT}}$ by using the factorization idea presented in 3.4.

Proposition 10. *Let $\overline{\mathbf{c}}$ be the output of the FDFB algorithm with input $[m]$. Then, the variance of the noise of $\overline{\mathbf{c}}$ verifies:*

$$Var(Err(\overline{\boldsymbol{c}})) \leq (||P_g - P_f||_2^2 + 1)\cdot \mathcal{E}_{BS} + (2\cdot ||P_g - P_f||_2^2 + 1)\cdot \mathcal{E}_{KS}$$

Proof. The result corresponds to the error of a ciphertext computed from a private functional bootstrapping (section 3.3) with a test vector that is obtained with a public functional bootstrapping, followed by a KeySwitch and a multiplication by a clear polynomial. Thus, we can compose the errors' formulas of each of these operations and get the final result. □

Proposition 11. *Let $\boldsymbol{c}$ be a TLWE ciphertext, and suppose that we differentiate $|\mathcal{M}|$ possible input values, the probability of error of the FDFB bootstrapping algorithm with input $\boldsymbol{c}$ verifies:*

$$P(Err(c)) = 1 - erf\left(\frac{1}{2\cdot|\mathcal{M}|\cdot\sqrt{V_c + V_r}\cdot\sqrt{2}}\right)$$

where $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in the bootstrapping algorithm.

Proof. For the first BlindRotate to succeed in computing the Heaviside function H, the noise of $\frac{\lfloor 2N\boldsymbol{c}\rceil}{2N}$ has to be smaller than $\frac{1}{4}$. Then, for the second BlindRotate to succeed and get the final result, the noise of $\frac{\lfloor 2N\boldsymbol{c}\rceil}{2N}$ has to be smaller than $\frac{1}{2|\mathcal{M}|}$. Since $|\mathcal{M}| \geq 2$, we just need to take into account the probability of error of the second BlindRotate. Finally, we get this probability of error thanks to the properties of *erf*. □

4.3 Full Domain Functional Bootstrapping–TOTA

Yan et al., [Yan+21] proposed this method to evaluate arbitrary functions over the torus using a functional bootstrapping. Let's consider a ciphertext $[m_1] = (\boldsymbol{a}, b = \boldsymbol{a}.\boldsymbol{s} + m_1 + e)$. Then, by dividing each coefficient of this ciphertext by 2, we get a ciphertext $[m_2] = (\frac{\boldsymbol{a}}{2}, \frac{\boldsymbol{a}}{2}.\boldsymbol{s} + m_2 + \frac{e}{2})$ where $m_2 = \frac{m_1}{2} + \frac{k}{2}$ with $k \in \{0,1\}$ and $\frac{m_1}{2} \in [0, \frac{1}{2}[$. Using the original bootstrapping algorithm, we compute $[\frac{\mathrm{sign}(m_2)}{4}]$ an encryption of $\frac{\mathrm{sign}(m_2)}{4} = \begin{cases} \frac{1}{4} & \text{if } k = 0 \\ -\frac{1}{4} & \text{if } k = 1 \end{cases}$. Then, $[m_2] - [\frac{\mathrm{sign}(m_2)}{4}] + (\mathbf{0}, \frac{1}{4})$ is an encryption of $\frac{m_1}{2}$.

For any function f, let's define $f_{(2)}$ such that $f_{(2)}(x) = f(2x)$. Since $\frac{m_1}{2} \in [0, \frac{1}{2}]$, we can compute $f_{(2)}(\frac{m_1}{2})$ with a single bootstrapping using the partial domain solution from 4.1, and $f_{(2)}(\frac{m_1}{2}) = f(m_1)$.

Thus, this method allows computing any function with only 2 bootstrappings. Keep in mind that the torus is actually discretized, so some noise and some loss of precision are introduced after dividing by 2 due to the rounding of the coefficients.

Proposition 12. *Let $\overline{\boldsymbol{c}}$ be the output of the TOTA functional bootstrapping algorithm for a given input. Then, the variance of the noise of $\overline{\boldsymbol{c}}$ verifies:*

$$Var(Err(\overline{\boldsymbol{c}})) \leq \mathcal{E}_{BS} + \mathcal{E}_{KS}$$

Proof. The algorithm ends with a functional bootstrapping which directly gives the result. □

Proposition 13. *Let $\boldsymbol{c}$ be a TLWE ciphertext, and suppose that we differentiate $|\mathcal{M}|$ possible input values, the probability of error of the TOTA algorithm with input $\boldsymbol{c}$ verifies:*

$$P(Err(\boldsymbol{c})) = 1 - erf(\frac{1}{4\sqrt{V_c + V_r} \cdot \sqrt{2}}) \cdot erf(\frac{1}{4 \cdot |\mathcal{M}| \cdot \sqrt{\frac{V_c}{4} + V_r + V_{sign}} \cdot \sqrt{2}})$$

where $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in the bootstrapping algorithm, and V_{sign} is the variance of the sign functional bootstrapping (i.e., $V_{sign} = \mathcal{E}_{BS} + \mathcal{E}_{KS}$).

Proof. We need to apply two BlindRotate over inputs. In order to compute the sign successfully, we need the noise of $\frac{\lfloor 2N\boldsymbol{c} \rceil}{2N}$ to be smaller than $\frac{1}{4}$. In order to compute the second BlindRotate successfully, we need the noise of $\frac{\lfloor N\boldsymbol{c} + 2N \cdot [\frac{sign}{4}] \rceil}{2N}$ to be smaller than $\frac{1}{4|\mathcal{M}|}$. Thus, the probability of success for the algorithm is the product of the probability of success for each BlindRotate. Knowing that the probability of error is the complementary to one of this product gives us the result. □

4.4 Full Domain Functional Bootstrapping with Composition

In this section, we present a novel method to compute any function using the full (discretized) torus as plaintext space. In this regard, it uses the same plaintext space as solutions presented in Sections 4.2 and 4.3.

4.4.1 Pseudo odd functions

We call pseudo odd function a function f that verifies $\forall x \in \mathbb{T}, f(-x - \frac{1}{|\mathcal{M}|}) = -f(x)$. We note $\lfloor x \rceil_{\mathcal{M}}$ the rounding function which discretizes the torus over $|\mathcal{M}|$ values, and $f_{\mathbb{T}}$ a pseudo odd function over the discretized torus.

Let h be the following function:

$$h: \begin{array}{ccc} [0, \frac{1}{2}[& \rightarrow & \mathbb{T} \\ x & \mapsto & \lfloor x \rceil_{\mathcal{M}} + \frac{1}{2|\mathcal{M}|} \end{array}$$

Then we can define a functional bootstrapping with an output function g_h as such:

$$g_h: x \mapsto \begin{cases} \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} + \frac{1}{2|\mathcal{M}|} & \text{if } x \in [\![0, N[\![\\ -\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - \frac{1}{2} - \frac{1}{2|\mathcal{M}|} & \text{if } x \in [\![-N, 0[\![\end{cases}$$

We now consider the restriction of $f_{\mathbb{T}}$ over positive values $[0,\frac{1}{2}[$. Then we can define $g_{f_{\mathbb{T}+}}$ as such:

$$g_{f_{\mathbb{T}+}}: x \mapsto \begin{cases} f_{\mathbb{T}}\left(\frac{x}{2N}\right) & \text{if } x \in [\![0,N[\![\\ -f_{\mathbb{T}}\left(\frac{x+N}{2N}\right) & \text{if } x \in [\![-N,0[\![\end{cases}$$

We can compose $g_{f_{\mathbb{T}+}}$ with $2Ng_h - \frac{N}{|\mathcal{M}|}$.

$$g_{f_{\mathbb{T}+}} \circ (2Ng_h - \frac{N}{|\mathcal{M}|}): x \mapsto \begin{cases} f_{\mathbb{T}}\left(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}}\right) & \text{if } \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} \in [0,\frac{1}{2}[\\ -f_{\mathbb{T}}\left(-\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - \frac{1}{|\mathcal{M}|}\right) & \text{if } \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} \in [-\frac{1}{2},0[\end{cases}$$

Considering that $f_{\mathbb{T}}$ is pseudo odd, we get:

$$\forall x \in \mathbb{T}, g_{f_{\mathbb{T}+}} \circ (2Ng_h - \frac{N}{|\mathcal{M}|})(x) = f_{\mathbb{T}}(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}})$$

Therefore $g_{f_{\mathbb{T}+}} \circ (2Ng_h - \frac{N}{|\mathcal{M}|})$ evaluates a LUT based on $f_{\mathbb{T}}$ for the whole discretized torus.

4.4.2 Pseudo even functions

We call pseudo even function a function f that verifies $\forall x \in \mathbb{T}, f(-x - \frac{1}{|\mathcal{M}|}) = f(x)$.

We note $f_{\mathbb{T}}$ a pseudo even function over the discretized torus.

We set h as:

$$h: \begin{array}{ccc} [0,\frac{1}{2}[& \rightarrow & \mathbb{T} \\ x & \mapsto & \lfloor x \rceil_{\mathcal{M}} + \frac{1}{4} + \frac{1}{2|\mathcal{M}|} \end{array}$$

Then we can define a functional bootstrapping with an output function g_h as such:

$$g_h: x \mapsto \begin{cases} \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} + \frac{1}{4} + \frac{1}{2|\mathcal{M}|} & \text{if } x \in [\![0,N[\![\\ -\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} + \frac{1}{4} - \frac{1}{2|\mathcal{M}|} & \text{if } x \in [\![-N,0[\![\end{cases}$$

We now can compose $g_{f_{\mathbb{T}+}}$ with $2Ng_h - \frac{N}{2} - \frac{N}{|\mathcal{M}|}$.

$$g_{f_{\mathbb{T}+}} \circ (2Ng_h - \frac{N}{2} - \frac{N}{|\mathcal{M}|}): x \mapsto \begin{cases} f_{\mathbb{T}}\left(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}}\right) & \text{if } \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} \in [0,\frac{1}{2}[\\ f_{\mathbb{T}}\left(-\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - \frac{1}{|\mathcal{M}|}\right) & \text{if } \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} \in [-\frac{1}{2},0[\end{cases}$$

Considering that $f_{\mathbb{T}}$ is pseudo even, we get:

$$\forall x \in \mathbb{T}, g_{f_{\mathbb{T}+}} \circ (2Ng_h - \frac{N}{|\mathcal{M}|})(x) = f_{\mathbb{T}}(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}})$$

Therefore, $g_{f_{\mathbb{T}+}} \circ (2Ng_h - \frac{N}{|\mathcal{M}|})$ is a LUT based on $f_{\mathbb{T}}$ over the whole discretized torus.

4.4.3 Any function

Any function $f_{\mathbb{T}}$ can be written as a sum of a pseudo even function and a pseudo odd function: $f_{\mathbb{T}}(x) = \frac{f_{\mathbb{T}}(x)+f_{\mathbb{T}}(-x-\frac{1}{|\mathcal{M}|})}{2} + \frac{f_{\mathbb{T}}(x)-f_{\mathbb{T}}(-x-\frac{1}{|\mathcal{M}|})}{2}$. Sections 4.4.1 and 4.4.2 showed we can build an homomorphic LUT based on any pseudo odd or pseudo even function with at most 2 functional bootstrapping operations. This means that we can build one over any kind of function with at most 4 functional bootstrapping operations. In practice, since both the pseudo odd and the pseudo even functions are evaluated on the same input, a multi-value functional bootstrapping (see Section 3.4) can be used to reduce the maximum amount of bootstrapping operations to 3. Besides, odd functions and even functions can be computed in a very similar way to their pseudo equivalent with only 2 bootstrappings. There are also a host of useful functions (sigmoid, monomial functions, trigonometric functions, identity, $\cdots$) which can be computed using only 2 bootstrapping operations because they are one sum away from an odd or even function.

Note that this solution is only suitable for precise arithmetic. Indeed, because of the negacyclic nature of the bootstrapping operation, we are actually composing discontinuous functions. This can lead to unexpected behaviors if the noise of the ciphertext is too big.

Proposition 14. *Let $\overline{\mathbf{c}}$ be the output of the composition functional bootstrapping algorithm for a given input. Then, the variance of the noise of $\overline{\mathbf{c}}$ verifies:*

$$Var(Err(\overline{\mathbf{c}})) \leq 2 \cdot (\mathcal{E}_{BS} + \mathcal{E}_{KS})$$

Proof. The result comes from the addition of two independent bootstrapped ciphertexts. □

Proposition 15. *Let $\mathbf{c}$ be a TLWE ciphertext, and suppose that we differentiate $|\mathcal{M}|$ possible input values. The probability of error of the composition functional bootstrapping algorithm with $\mathbf{c}$ verifies:*

$$P(Err(\boldsymbol{c})) = 1 - erf\left(\frac{1}{2 \cdot |\mathcal{M}| \cdot \sqrt{V_c + V_r} \cdot \sqrt{2}}\right) \cdot \left(erf\left(\frac{1}{2 \cdot |\mathcal{M}| \cdot \sqrt{V_f + V_r} \cdot \sqrt{2}}\right)\right)^2$$

where $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in the bootstrapping algorithm, and V_f is the variance of the result of the first functional bootstrapping (i.e., $V_f = \mathcal{E}_{BS} + \mathcal{E}_{KS}$).

Proof. The proof is similar to the proof of Proposition 13. □

5 Look-Up-Tables over Multiple Ciphertexts

In section 4, we discussed several functional bootstrapping methods that take as input one ciphertext. These methods have a limited plaintext space and precision, and allow evaluating look-up tables with a size bounded by the degree of the used cyclotomic polynomial (N). In addition, these methods are not suited for computing a LUT for a multivariate function f that takes as inputs two or more ciphertexts. In order to overcome these issues, we describe in this section a method for computing functions using multiple ciphertexts as inputs.

Our proposed solution improves the results of Guimarães et al., [GBA21]. They, themselves, generalize the ideas of Boura et al. [BST19] and discuss two methods for homomorphic computation with digits: a tree-based approach and a chaining approach. We expand on the chaining method in order to obtain any function through its use as opposed to the subset of function previously allowed.

Subsequently, we use this method to apply a LUT to a *single* message decomposed over *multiple* ciphertexts. That is, we decompose each plaintext into several digits in a certain base B and encrypt these digits separately. Decomposition allows working with a larger plaintext space $\mathcal{I}$ while using an acceptable parameters set for an efficient computation.

In this section, we first *review* the tree-based method and then *improve* the chaining method to make it fit any function. We show how those methods can be used as building blocks in order to compute additions and multiplications of messages decomposed over multiple ciphertexts. We then show how to compute the ReLU function over a single, decomposed, plaintext. The choice of ReLU as a worthy application of our novel method was made because it is the most used activation function in modern convolutional neural networks.

5.1 Tree-based Method

We consider d TLWE ciphertexts $(\boldsymbol{c}_0,\ldots,\boldsymbol{c}_{d-1})$ encrypting the messages $(m_0,\ldots,m_{d-1})$ over half of the torus and $B\in\mathbb{N}$, such that each ciphertext $\boldsymbol{c}_i$ corresponds to an encryption of $m_i\in[\![0,B-1]\!]$. We denote by $f:[\![0,B-1]\!]^d\to[\![0,B-1]\!]$ our target function and by g the bijection:

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

We encode the LUT for f in B^{d-1} TRLWE ciphertexts. Each ciphertext encrypts a polynomial P_i where:

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

Then, we apply the BlindRotate algorithm to $\boldsymbol{c}_{d-1}$ and each $\mathsf{TRLWE}(P_i)$, and use the SampleExtract algorithm to extract the first coefficient of the result. We end up with B^{d-1} TLWE ciphertexts each encrypting a message $f\circ g^{-1}(m_{d-1}\cdot B^{d-1}+i)$ for $i\in[\![0,B^{d-1}-1]\!]$. Thanks to TLWE to TRLWE keyswitching, we batch them into B^{d-2} TRLWE ciphertexts corresponding to the LUT of h where:

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

We iterate this operation until getting only one TLWE ciphertext encrypting $f(m_0,\ldots,m_{d-1})$. Since a function from $[\![0,B-1]\!]^d$ to $[\![0,B-1]\!]^k$ can be decomposed in k functions from $[\![0,B-1]\!]^d$ to $[\![0,B-1]\!]$, we can actually build any function between any inputs, once they are decomposed in base B then encrypted.

Note that the BlindRotate algorithm is costly and we have to call it $\sum_{i=0}^{d-1}B^i=\frac{B^d-1}{B-1}$ times. Fortunately, we can make it faster by encoding the first LUTs in plaintext polynomials rather than TRLWE ciphertexts. Then, we use the multi-value bootstrapping given in [CIM19] to compute only one bootstrapping instead of B^{d-1} in the first step of the algorithm. Thus we end-up by running $1+\sum_{i=0}^{d-2}B^i=1+\frac{B^{d-1}-1}{B-1}$ BlindRotate.

Proposition 16. *Let $\overline{\mathbf{c}}$ be the output of the tree-based functional bootstrapping algorithm for a given input on d digits. Then, if we don't use the multi-value bootstrapping for the first level of the tree, the variance of the noise of $\overline{\mathbf{c}}$ will verify:*

$$Var(Err(\overline{\mathbf{c}}))\leq d\cdot(\mathcal{E}_{BS}+\mathcal{E}_{KS})$$

If we use the multi-value bootstrapping with polynomials P_i we get:

$$Var(Err(\overline{\mathbf{c}}))\leq(d-1+\max(||P_i||_2^2))\cdot\mathcal{E}_{BS}+d\cdot\mathcal{E}_{KS}$$

Proof. The result comes from the composition of the formulas for multi-value functional bootstrapping, keyswitching, and private functional bootstrapping. ☐

Proposition 17. *Let $(\boldsymbol{c}_i)_{i\in[\![1,d]\!]}$ be d TLWE ciphertexts corresponding to d digits of a plaintext message. Suppose that we differentiate $|\mathcal{M}|$ possible input values, the probability of error of the tree-based bootstrapping algorithm with inputs $(\boldsymbol{c_i})_{i\in[\![1,d]\!]}$ verifies:*

$$P(Err((\boldsymbol{c}_i)_{i\in[\![1,d]\!]})) = 1 - \prod_{i=1}^{d} erf(\frac{1}{4\cdot|\mathcal{M}|\cdot\sqrt{V_{c_i}+V_r}\cdot\sqrt{2}})$$

where $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in the bootstrapping algorithm.

Proof. The result comes from the fact that for each i, $\boldsymbol{c}_i$ must have a noise low enough to allow for a successful BlindRotate. ☐

5.2 Chaining Method

The chaining method has a much lower complexity and a lower error growth than the tree-based method but, as presented in [GBA21], works only for a more restricted set of functions.

We consider n TLWE ciphertexts $(\boldsymbol{c}_0,\ldots,\boldsymbol{c}_{n-1})$ encrypting the messages $(m_0,\ldots,m_{n-1})$ respectively and denote by $LC(a,b)$ *any* linear combination of a and b. Given some functions $(f_i)_{i\in[\![0,n-1]\!]}$ so that $f_i:[\![0,B-1]\!]\to[\![0,B-1]\!]$, we can build a function $f:[\![0,B-1]\!]^n\to[\![0,B-1]\!]$ following Algorithm 4. Each f_i can be implemented in the homomorphic domain using any functional bootstrapping method described in Section 4. The result of this algorithm has the same noise as a simple functional bootstrapping, thus much less than the noise output of the tree method.

Algorithm 4 Chaining method

Input: A vector $(\boldsymbol{c}_0,\ldots,\boldsymbol{c}_{n-1})$ of TLWE ciphertexts encrypting the vector of messages $(m_0,\ldots,m_{n-1})$.

Output: A ciphertext encrypting $f(m_0,\ldots,m_{n-1})$. f is defined here by the linear combinations chosen at every step and the different single-input functions f_i.

$\overline{\boldsymbol{c}}_0 \leftarrow f_0(\boldsymbol{c}_0)$
for $i\in[\![0,n-2]\!]$ **do**
  $\overline{\boldsymbol{c}}_{i+1} \leftarrow f_{i+1}(LC(\overline{\boldsymbol{c}}_i,\boldsymbol{c}_{i+1}))$
return $\overline{\boldsymbol{c}}_{\boldsymbol{n-1}}$

Most functions cannot be computed in such a simplistic way, which greatly restricts its use even though it can be effective for functions with carry-like logic as stated in [GBA21].

Generalization. It is possible to build any function f using a similar method. We introduce the function g such that:

$$g: \begin{array}{ccc} [\![0,B-1]\!]^2 & \to & [\![0,B^2-1]\!] \\ (a_0,a_1) & \mapsto & a_0+a_1\cdot B \end{array}$$

That function is a bijection, which means that if a ciphertext can hold any message in $[\![0,B^2-1]\!]$, then we can compute any function of two ciphertexts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ by applying one functional bootstrapping over $g(\boldsymbol{c}_1,\boldsymbol{c}_2)$.

Note that when using base 2, we can easily build any logic gate with this method. We can then build a circuit with these gates for any functions. The same idea works for any base B. However, this generalization comes at the cost of multiple bits of padding and the conception of the proper circuit.

Proposition 18. *Let $\overline{\mathbf{c}}$ be the output of the chaining functional bootstrapping algorithm for given encrypted d digits. Then, the variance of the error of $\overline{\mathbf{c}}$ follows the same formula as the last functional bootstrapping method used in the chain.*

Proof. We get the result by applying the noise formula associated to the last functional bootstrapping in the chain and by noticing that it does not depend on the noise of the input. □

The probability of error is highly dependent on the choice of: the encoded LUT in the functional bootstrapping applied to each digit, the linear combinations between the inputs and outputs of the chained bootstrappings, and the structure of the circuit corresponding to the target function. Thus, a general formula cannot be given.

5.3 Addition

We expect additions of two messages to be computed in linear time with respect to the number of digits of each message. Thus the tree-based method is ill-suited for this operation, since the tree-based method computing time grows exponentially with the number of digits used as inputs. Meanwhile, the chaining method is not exactly adapted to this operation if applied directly. Nonetheless, we show that we can still use any of the two methods to compute the addition effectively.

Let $m_1 = \sum_{i=0}^{n} m_{1,i} \cdot B^i$ and $m_2 = \sum_{i=0}^{n} m_{2,i} \cdot B^i$ be two messages expressed in base B. For each pair (i,j), let $\boldsymbol{c}_{i,j}$ be the ciphertext encrypting the message $m_{i,j}$. We define $\boldsymbol{c}_i = (\boldsymbol{c}_{i,0},\ldots,\boldsymbol{c}_{i,n})$ as the vector of ciphertexts encrypting m_i in base B. Finally, we denote by h the half adder function, and by f the full adder one:

$$h\colon \begin{array}{ccc} [\![0,B-1]\!]^2 & \to & [\![0,B-1]\!]^2 \\ (a,b) & \mapsto & ((a+b)[B], \lfloor (a+b)/B \rfloor) \end{array}$$

$$f\colon \begin{array}{ccc} [\![0,B-1]\!]^2 \times \{0,1\} & \to & [\![0,B-1]\!]^2 \\ (a,b,c) & \mapsto & ((a+b+c)[B], \lfloor (a+b+c)/B \rfloor) \end{array}$$

These two functions are the only requirements to build the addition operation. But, in order to be able to create those two adders, we need to create the following sub-functions:

$$mod\colon \begin{array}{ccc} [\![0,2B-1]\!] & \to & [\![0,B-1]\!] \\ x & \mapsto & x[B] \end{array}$$

$$carry\colon \begin{array}{ccc} [\![0,2B-1]\!] & \to & \{0,1\} \\ x & \mapsto & \lfloor x/B \rfloor \end{array}$$

We can use either the tree-based method or the chaining method to compute *mod* or *carry* functions. The chaining method needs one bit of padding to work, while the tree-based method is slower, especially for the full adder which is a three inputs function. Finally, we present Algorithm 5 for computing addition between two vectors of ciphertexts.

The time complexity of Algorithm 5 is linear with respect to the number of digits of the entries. The noise of each output ciphertext is the same as the noise of a simple bootstrapping if we use the chaining method for computing the sub-functions *mod* and *carry*. Meanwhile, with the tree-based method, we end-up with the noise of a simple bootstrapping followed by two BlindRotate.

Algorithm 5 Addition

Input: Two vectors of ciphertexts $\boldsymbol{c}_1 = (\boldsymbol{c}_{1,i})_{i\in[\![0,n-1]\!]}$ and $\boldsymbol{c}_2 = (\boldsymbol{c}_{2,i})_{i\in[\![0,m-1]\!]}$ encrypting two messages m_1 and m_2 written in base B. We suppose here that $n \geq m$.
Output: An encryption of $m_1 + m_2$ in base B.
$(\overline{\boldsymbol{c}}_{1,0}, \overline{\boldsymbol{c}}_{2,0}) \leftarrow h(\boldsymbol{c}_{1,0}, \boldsymbol{c}_{2,0})$
for $i \in [\![0, m-2]\!]$ **do**
  $(\overline{\boldsymbol{c}}_{1,i+1}, \overline{\boldsymbol{c}}_{2,i+1}) \leftarrow f(\boldsymbol{c}_{1,i+1}, \boldsymbol{c}_{2,i+1}, \overline{\boldsymbol{c}}_{2,i})$
for $i \in [\![m-1, n-2]\!]$ **do**
  $(\overline{\boldsymbol{c}}_{1,i+1}, \overline{\boldsymbol{c}}_{2,i+1}) \leftarrow h(\boldsymbol{c}_{1,i+1}, \overline{\boldsymbol{c}}_{2,i})$
return $(\overline{\boldsymbol{c}}_{1,0}, \ldots, \overline{\boldsymbol{c}}_{1,n-1}, \overline{\boldsymbol{c}}_{2,n-1})$

5.4 Multiplication

As we expected linear computation time to be achievable for the homomorphic addition, we expect to achieve quadratic time complexity for homomorphic multiplication. Let m_1 and m_2 be two messages and $\boldsymbol{c}_1 = (\boldsymbol{c}_{1,i})_{i\in[\![0,n-1]\!]}$ and $\boldsymbol{c}_2 = (\boldsymbol{c}_{2,i})_{i\in[\![0,m-1]\!]}$ be their encryption in base B. In order to evaluate $m_1 \cdot m_2$ in the encrypted domain, we first multiply each digit of m_1 by each digit of m_2. Then, we have just to add the obtained elements properly using half and full adders to get the final result.

Since we have already introduced homomorphic adders, we only need to describe how to multiply two digits. Given two messages a and b in $[\![0, B-1]\!]$, we need to compute $a \cdot b[B]$ and $a \cdot b/B$ in the encrypted domain. If we use the tree-base method, we can compute both functions with three LUTs since both functions will use the same selector in the first step. Otherwise, we can also use the generalized chaining method to compute both needed functions using two LUTs, but this method comes at the cost of using multiple bits of padding.

We denote by $\mathsf{MultDigits}(\boldsymbol{c}_a, \boldsymbol{c}_b)$ a method for computing $a \cdot b[B]$. In the same way, we denote by $\mathsf{CarryMult}(\boldsymbol{c}_a, \boldsymbol{c}_b)$ a method for computing $a \cdot b/B$. Then the multiplication of m_1 and m_2 can be done with Algorithm 6.

Algorithm 6 Multiplication

Input: Two vectors of ciphertexts $\boldsymbol{c}_1 = (\boldsymbol{c}_{1,i})_{i\in[\![0,n-1]\!]}$ and $\boldsymbol{c}_2 = (\boldsymbol{c}_{2,i})_{i\in[\![0,m-1]\!]}$ encrypting two messages m_1 and m_2 written in base B.
Output: An encryption $\overline{\boldsymbol{c}} = (\overline{\boldsymbol{c}}_i)_{i\in[\![0,n+m-1]\!]}$ of $m_1 \cdot m_2$ in base B.
for $i \in [\![0, n+m-1]\!]$ **do**
  $\mathrm{SubMul}_i \leftarrow$ empty vector
for $i \in [\![[0, n-1]\!]$ **do**
  for $j \in [\![0, m-1]\!]$ **do**
    Put $\mathsf{MultDigits}(\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,j})$ in vector SubMul_{i+j}
    Put $\mathsf{CarryMult}(\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,j})$ in vector SubMul_{i+j+1}
$\overline{\boldsymbol{c}}_0 \leftarrow \mathrm{SubMul}_0[0]$
for $i \in [\![1, n+m-1]\!]$ **do**
  $\overline{\boldsymbol{c}}_i \leftarrow (\sum_{j=0}^{\mathrm{size}(\mathrm{SubMul}_i)-1} \mathrm{SubMul}_i[j])[B]$ using adders
  Put the carries in SubMul_{i+1}
return $(\overline{\boldsymbol{c}}_0, \ldots, \overline{\boldsymbol{c}}_{n+m-1})$

The time complexity of Algorithm 6 is quadratic with respect to the number of digits of the entries. The noise of the outputs is similar to the noise of the adder sub-functions.

5.5 ReLU

In this section, we describe how to avoid using the tree-based method, as it is, for the implementation of the ReLU activation function. Let's consider $m=\sum_{i=0}^{n} m_i \cdot B^i$ a message written using radix complement representation in base B, and $(\boldsymbol{c}_i)_{i\in[\![0,n]\!]} = (\mathrm{TLWE}_{\boldsymbol{s}}(m_i))_{i\in[\![0,n]\!]}$.

In order to use the tree-based method to evaluate intermediate functions on each encrypted digit, we use a functional bootstrapping to create a selector S from $\boldsymbol{c}_n$ that encrypts the torus element 0 if $0 \le m_n < \frac{B}{2}$ and $\frac{1}{4}$ if $\frac{B}{2} \le m_n < B$. Note that $(0 \le m_n < \frac{B}{2}) \iff (m \ge 0)$, so the value of S depends on the sign of m. Then, for each $\boldsymbol{c}_i$, we create using keyswitching a TRLWE ciphertext $\mathrm{LUT}(\boldsymbol{c}_i)$ so that for $j \in [\![0, \frac{N}{2}-1]\!]$, $\mathsf{SampleExtract}(\mathrm{LUT}(\boldsymbol{c}_i), j)$ is an encryption of m_i, and for $j \in [\![\frac{N}{2}, N-1]\!]$, $\mathsf{SampleExtract}(\mathrm{LUT}(\boldsymbol{c}_i), j)$ is an encryption of 0. Then, $\mathsf{SampleExtract}(\mathsf{BlindRotate}(S, \mathrm{LUT}(\boldsymbol{c}_i), 0)$ outputs:

$$\overline{c_i} = \begin{cases} \mathrm{TLWE}(0,s) & \text{if } m<0 \\ \mathrm{TLWE}(m_i,s) & \text{if } m \ge 0 \end{cases}$$

Thus, $(\overline{\boldsymbol{c}}_i)_{i\in[\![0,n]\!]}$ encrypts $\mathsf{ReLU}(m)$ using radix complement representation in base B.

Otherwise, we can compute the ReLU function using the chaining method. Then, each ciphertext has to encrypt a value in $[\![0,2B[\![$. First, let's compute a selector S from c_n such that:

$$S = \begin{cases} \mathrm{TLWE}(0,s) & \text{if } m \ge 0 \\ \mathrm{TLWE}(B,s) & \text{if } m<0 \end{cases}$$

Then, let's define:

$$f : \begin{array}{ccc} [\![0,2B-1]\!] & \to & [\![0,2B-1]\!] \\ x & \mapsto & \begin{cases} x & \text{if } x<B \\ 0 & \text{if } x \ge B \end{cases} \end{array}$$

This function can be computed with one functional bootstrapping. For each $\boldsymbol{c}_i$, we compute $\overline{\boldsymbol{c}}_i = f(\boldsymbol{c}_i + S)$. We obtain $(\overline{\boldsymbol{c}}_i)_{i\in[\![0,n]\!]}$ an encryption using radix complement representation in base B of $\mathsf{ReLU}(m)$.

6 Experimental Results

In this section, we compare time and accuracy performances for each of the functional bootstrapping presented above.

Parameters. We considered a wide panel of parameters' sets with either $\lambda = 80$ or 120 bits of security. Considering that λ only depends on the parameters n, N and $\sigma_{\min}$ which is the standard deviation of fresh ciphertexts, we set those parameters as shown in Table 1. We set the parameters t and B_{KS} relative to key switching to $t=3$ and $B_{KS}=128$. This way, KeySwitch operations are fast enough to be negligible compared to bootstrappings and the resulting noise has a very low impact compared to the other sources of noise. The other parameters are chosen to give a good representation of the ability of each method and can be seen with the results of the experiment in Tables 2, 3, 4, and 5 (which are at the end of the paper).

Accuracy. In the tables mentioned earlier, we computed the probability of error ϵ of each method for every set of parameters considered, allowing for a comparison between them. Note that the probability of error of each method does not depend on the function applied by the bootstrapping except for FDFB. In this particular case, we gave the probability of error using the functions Id and ReLU as well as the worst case. The experiment shows that for any given set of parameters, the probability of error is identical between TOTA and the partial domain method, or slightly in favor of the latter. Meanwhile, the composition method gets much better results than any other method in every case. In the case of FDFB, we can see that the smaller the parameter l is, the worst it is compared to the others. On the opposite, when l becomes bigger, its results become much better and even compete with the composition method. In addition, the choice of the function has a big impact on ϵ, and in simple cases such as the ReLU and Id function, even the worst set of parameters get similar result to the partial domain method and TOTA.

Time performance. In every case, the speed of each method can be closely approximated by the speed of one simple bootstrapping multiplied by the number of bootstrapping needed. This result in the partial domain method being the fastest with only 1 bootstrapping needed. Then, TOTA is slightly faster than FDFB as it requires less key switching operations. As far as the composition method is concerned, the number of bootstrapping depends on the function evaluated. Thus, for a simple function such as the absolute value, its speed is identical to that of the partial domain method. Meanwhile, the ReLU function needs 3 bootstrappings which leads to it being about $\frac{3}{2}$ times slower than TOTA and FDFB.

7 Conclusion

Through the use of several bootstrapping operations and - in some cases - additional operations, every full domain method (Sections 4.2, 4.3 and 4.4) adds some output noise when compared to the simpler and quicker partial domain method (Section 4.1). The question is: does a larger initial plaintext space make up for the added noise and computation time? Table 2 and Table 3 shows us that the Yan et al., [Yan+21] (TOTA) method is both less accurate and twice as time-consuming than the partial domain method. Kluczniak and Schild's [KS21] (FDFB) method, gets a better accuracy for well chosen parameters but is still twice as time-consuming as the partial domain method. Our novel composition method (Section 4.4) is more accurate than any of the previously mentioned methods, however thrice as time consuming as the partial-domain method. As for our digit-decomposition method (Section 5), it allows for an arbitrary precision, though with a corresponding running time always much higher than the partial domain solution.
Given these experimental measures, our recommendations on the use of these functional bootstrapping methods are the following, given specific applicative scenarios:

- **Precise integer arithmetic above all else.** In some cases, precision is the only criteria that matters. Then, **our generalized digit-decomposition functional bootstrapping method** is the appropriate choice as it is the *only* method with unbounded precision for functional bootstrapping computation of *any* function in the literature.
- **Efficient approximate or precise integer arithmetic.** In the case where we need either an approximate or a precise arithmetic computation in a limited amount of time, the **partial domain method** is an obvious choice. Its precision is only constantly toped by our Composition method, but the speed difference is the decisive factor here.
- **Efficient precise modular arithmetic.** There is a case where one wishes to use modular arithmetic instead of integer arithmetic. In this case, the partial domain method cannot be used as plaintexts are encoded on only half of the torus which is not an additive group. In this case one of the full domain methods must be used. If the computation must be precise then **our novel composition method** is the most precise among the options.
- **Efficient approximate modular arithmetic.** In the case where the arithmetic is modular but the computation is approximate due to large noises in ciphertexts, the composition method should be avoided as its behavior becomes unpredictable. Therefore the preferred option becomes **FDFB** [KS21].

Furthermore, the operators presented in this paper provide key building blocks for enabling advanced deep learning functions over encrypted data.

References

[Bou+20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. "CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes". In: *Journal of Mathematical Cryptology* 14.1 (1Jan. 2020), pp. 316–338. DOI: https://doi.org/10.1515/jmc-2019-0026. URL: https://www.degruyter.com/view/journals/jmc/14/1/article-p316.xml.

[Bou+19] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. "Simulating Homomorphic Evaluation of Deep Learning Predictions". In: *Cyber Security Cryptography and Machine Learning*. Ed. by Shlomi Dolev, Danny Hendler, Sachin Lodha, and Moti Yung. Cham: Springer International Publishing, 2019, pp. 212–230.

[Bou+18] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. "Fast Homomorphic Evaluation of Deep Discretized Neural Networks". In: *Proceedings of CRYPTO 2018*. Springer, 2018.

[BST19] Florian Bourse, Olivier Sanders, and Jacques Traoré. *Improved Secure Integer Comparison via Homomorphic Encryption*. Cryptology ePrint Archive, Report 2019/427. https://ia.cr/2019/427. 2019.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886. ISBN: 978-3-642-32009-5.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. "New Techniques for Multi-value Input Homomorphic Evaluation and Applications". In: *Topics in Cryptology – CT-RSA 2019*. Ed. by Mitsuru Matsui. Cham: Springer International Publishing, 2019, pp. 106–126. ISBN: 978-3-030-12612-4.

[Cha+19] Herve Chabanne, Roch Lescuyer, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. "Recognition Over Encrypted Faces: 4th International Conference, MSPN 2018, Paris, France". In: 2019.

[Cha+17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. *Privacy-Preserving Classification on Deep Neural Network*. Cryptology ePrint Archive, Report 2017/035. 2017.

[Che+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: (2017). Ed. by Tsuyoshi Takagi and Thomas Peyrin.

[CKP19] Jung Hee Cheon, Duhyeong Kim, and Jai Hyun Park. "Towards a Practical Clustering Analysis over Encrypted Data". In: *IACR Cryptology ePrint Archive* (2019).

[Chi+16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 3–33. ISBN: 978-3-662-53887-6.

[Chi+17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE". In: *ASIACRYPT*. 2017.

[Chi+] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library*. URL: https://tfhe.github.io/tfhe/.

[Chi+19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *Journal of Cryptology* 33 (Apr. 2019). DOI: 10.1007/s00145-019-09319-x.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks". In: *Cyber Security Cryptography and Machine Learning*. Ed. by Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann. Cham: Springer International Publishing, 2021, pp. 1–19. ISBN: 978-3-030-78086-9.

[Chi+21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. *Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE*. Cryptology ePrint Archive, Report 2021/729. https://ia.cr/2021/729. 2021.

[DM15] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 617–640. ISBN: 978-3-662-46800-5.

[FV12] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. https://ia.cr/2012/144. 2012.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. "Revisiting the functional bootstrap in TFHE". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (Feb. 2021), pp. 229–253. DOI: 10.46586/tches.v2021.i2.229-253. URL: https://tches.iacr.org/index.php/TCHES/article/view/8793.

[ISZ19] M. Izabachène, R. Sirdey, and M. Zuber. "Practical Fully Homomorphic Encryption for Fully Masked Neural Networks". In: *Cryptology and Network Security - 18th International Conference, CANS 2019, Proceedings*. Vol. 11829. Lecture Notes in Computer Science. Springer, 2019, pp. 24–36.

[JA18] Angela Jäschke and Frederik Armknecht. "Unsupervised Machine Learning on Encrypted Data". In: *IACR Cryptology ePrint Archive* (2018).

[KS21] Kamil Kluczniak and Leonard Schild. *FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2021/1135. https://ia.cr/2021/1135. 2021.

[Lou+20] Qian Lou, Bo Feng, Geoffrey Charles Fox, and Lei Jiang. "Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9193–9202. URL: https://proceedings.neurips.cc/paper/2020/file/685ac8cadc1be5ac98da9556bc1c8d9e-Paper.pdf.

[Mad+21] Abbass Madi, Oana Stan, Aurélien Mayoue, Arnaud Grivet-Sébert, Cédric Gouy-Pailler, and Renaud Sirdey. "A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing". In: *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*. 2021, pp. 1–8. DOI: 10.1109/RDAAPS48126.2021.9452005.

[Nan+19] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. "Towards Deep Neural Network Training on Encrypted Data". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 40–48. DOI: 10.1109/CVPRW.2019.00011.

[Séb+21] Arnaud Grivet Sébert, Rafael Pinot, Martin Zuber, Cédric Gouy-Pailler, and Renaud Sirdey. "SPEED: secure, PrivatE, and efficient deep learning". In: *Mach. Learn.* 110.4 (2021), pp. 675–694. DOI: 10.1007/s10994-021-05970-3. URL: https://doi.org/10.1007/s10994-021-05970-3.

[Xie+14] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. "Crypto-Nets: Neural Networks over Encrypted Data". In: *CoRR* (2014).

[Yan+21] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. *TOTA: Fully Homomorphic Encryption with Smaller Parameters and Stronger Security.* Cryptology ePrint Archive, Report 2021/1347. https://ia.cr/2021/1347. 2021.

[ZCS20a] Martin Zuber, Sergiu Carpov, and Renaud Sirdey. "Towards Real-Time Hidden Speaker Recognition by Means of Fully Homomorphic Encryption". In: *Information and Communications Security.* Ed. by Weizhi Meng, Dieter Gollmann, Christian D. Jensen, and Jianying Zhou. Cham: Springer International Publishing, 2020, pp. 403–421. ISBN: 978-3-030-61078-4.

[ZCS20b] Martin Zuber, Sergiu Carpov, and Renaud Sirdey. "Towards real-time hidden speaker recognition by means of fully homomorphic encryption". In: *International Conference on Information and Communications Security.* Springer. 2020, pp. 403–421.

[ZS21] Martin Zuber and Renaud Sirdey. "Efficient homomorphic evaluation of k-NN classifiers". In: *Proc. Priv. Enhancing Technol.* 2021.2 (2021), pp. 111–129. DOI: 10.2478/popets-2021-0020. URL: https://doi.org/10.2478/popets-2021-0020.

A Multi-value Bootstrapping

We remind that any test polynomial for a $\mathrm{LUT}(f_i)$ can be factorized as:

$$\mathrm{LUT}(f_i) = \sum_{i=0}^{N-1} \alpha_i X^i = v_0 \cdot v_i \; mod[X^N + 1]$$

$$v_0 = \frac{1}{2}.(1 + \cdots + X^{N-1})$$

$$v_i = \alpha_0 + \alpha_{N-1} + (\alpha_1 - \alpha_0).X + \cdots + (\alpha_{N-1} - \alpha_0).X^{N-1}$$

Algorithm 7 Multi-value bootstrapping

Input: a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s} \to \boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$, k LUTs s.t. $\mathrm{LUT}(f_i) = v_0.v_i, \forall i \in [\![1,k]\!]$

Output: a list of k TLWE samples $\overline{\boldsymbol{c}}_{\boldsymbol{i}} \in \mathrm{TLWE}_{\boldsymbol{s}}(f_i(\frac{\phi(\bar{\boldsymbol{a}},\bar{b})}{2N}))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
2: Let $testv := v_0$
3: $\mathrm{ACC} \leftarrow \mathsf{BlindRotate}((\boldsymbol{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
4: **for** $i \leftarrow 1$ to k **do**
5: $\quad \mathrm{ACC}_i := \mathrm{ACC} \cdot v_i$
6: $\quad \overline{\boldsymbol{c}}_{\boldsymbol{i}} = \mathsf{SampleExtract}(\mathrm{ACC}_i)$
7: $\quad$ return $\mathsf{KeySwitch}_{\mathsf{s'} \to \mathsf{s}}(\overline{\boldsymbol{c}}_{\boldsymbol{i}})$

Proposition 19. *Let $\overline{\boldsymbol{c}}_{\boldsymbol{i}}$ be the i^{th} output of the multi-value functional bootstrapping algorithm with input $\boldsymbol{c}$. Then, the variance of the noise of $\overline{\boldsymbol{c}}_{\boldsymbol{i}}$ verifies:*

$$Var(Err(\overline{\boldsymbol{c}}_{\boldsymbol{i}})) \leq ||v_i||_2^2 \cdot \mathcal{E}_{BS} + \mathcal{E}_{KS}$$

Proof. The result comes from the fact that we simply multiply the result of a functional bootstrapping with a clear polynomial. □

Proposition 20. *Given $\boldsymbol{c}$ a TLWE ciphertext, and suppose that we discretize the torus over $|\mathcal{M}|$ values, the probability of error of the multi-value bootstrapping algorithm with $\boldsymbol{c}$ as an input verifies:*

$$P(Err(c)) = 1 - erf\left(\frac{1}{2 \cdot |\mathcal{M}| \sqrt{V_c + V_r} \cdot \sqrt{2}}\right)$$

where $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in line 1 of Algorithm 7.

Proof. The multiplication by a plaintext polynomial has no impact on the probability of error. Thus, the probability of error is the same as a simple functional bootstrapping. □

Table 1: Parameters and security

n	N	$\sigma_{\min}$	λ
1024	1024	$7.8e^{-09}$	120
900	1024	$8.4e^{-08}$	120
800	1024	$5.9e^{-07}$	120
700	1024	$4e^{-06}$	120
600	1024	$2.8e^{-05}$	120
1024	1024	$1.05e^{-11}$	80
900	1024	$5e^{-11}$	80
800	1024	$3.5e^{-10}$	80
700	1024	$5.5e^{-09}$	80
600	1024	$9.4e^{-08}$	80
500	1024	$1.5e^{-06}$	80

Table 2: Parameters and probability of error for half Torus method

n	N	l	Bgbit	$\sigma_{\min}$	ϵ	$\lvert\mathcal{M}\rvert$	λ	time (ms)
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-10}$	16	120	81.4
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-35}$	8	120	80.8
1024	1024	4	5	$7.8e^{-09}$	$\leq 2^{-37}$	8	120	92.0
1024	1024	5	5	$7.8e^{-09}$	$\leq 2^{-37}$	8	120	105.2
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-132}$	4	120	80.8
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-10}$	16	120	92.5
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-25}$	8	120	82.5
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-34}$	8	120	92.0
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-93}$	4	120	81.8
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-130}$	4	120	92.3
800	1024	9	2	$5.9e^{-07}$	$\leq 2^{-20}$	8	120	119.7
800	1024	4	4	$5.9e^{-07}$	$\leq 2^{-15}$	4	120	72.3
800	1024	6	3	$5.9e^{-07}$	$\leq 2^{-41}$	4	120	91.8
800	1024	13	2	$5.9e^{-07}$	$\leq 2^{-65}$	4	120	156.5
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-10}$	16	80	80.0
1024	1024	4	7	$1.05e^{-11}$	$\leq 2^{-10}$	16	80	93.5
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-37}$	8	80	80.7
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-142}$	4	80	81.3
900	1024	3	7	$5e^{-11}$	$\leq 2^{-12}$	16	80	70.9
900	1024	4	7	$5e^{-11}$	$\leq 2^{-12}$	16	80	81.4
900	1024	3	7	$5e^{-11}$	$\leq 2^{-42}$	8	80	70.6
900	1024	3	7	$5e^{-11}$	$\leq 2^{-161}$	4	80	70.7
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-13}$	16	80	63.0
800	1024	4	7	$3.5e^{-10}$	$\leq 2^{-13}$	16	80	72.4
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-47}$	8	80	63.4
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-180}$	4	80	64.3
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-14}$	16	80	54.9
700	1024	12	2	$5.5e^{-09}$	$\leq 2^{-15}$	16	80	129.7
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-51}$	8	80	55.7
700	1024	4	6	$5.5e^{-09}$	$\leq 2^{-53}$	8	80	63.3
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-198}$	4	80	54.9
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-17}$	8	80	48.1
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-33}$	8	80	55.1
600	1024	13	2	$9.4e^{-08}$	$\leq 2^{-59}$	8	80	119.5
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-62}$	4	80	47.5
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-125}$	4	80	55.1

Table 3: Parameters and probability of error for TOTA

n	N	l	Bgbit	$\sigma_{\min}$	ϵ	$\mathcal{M}$	λ	time (ms)
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-10}$	16	120	160.9
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-34}$	8	120	162.2
1024	1024	4	5	$7.8e^{-09}$	$\leq 2^{-36}$	8	120	184.7
1024	1024	5	5	$7.8e^{-09}$	$\leq 2^{-37}$	8	120	209.4
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-129}$	4	120	161.1
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-9}$	16	120	183.7
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-23}$	8	120	164.1
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-33}$	8	120	183.4
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-84}$	4	120	162.9
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-124}$	4	120	242.6
800	1024	9	2	$5.9e^{-07}$	$\leq 2^{-18}$	8	120	238.8
800	1024	4	4	$5.9e^{-07}$	$\leq 2^{-13}$	4	120	144.3
800	1024	6	3	$5.9e^{-07}$	$\leq 2^{-35}$	4	120	183.0
800	1024	13	2	$5.9e^{-07}$	$\leq 2^{-56}$	4	120	312.5
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-10}$	16	80	159.9
1024	1024	4	7	$1.05e^{-11}$	$\leq 2^{-10}$	16	80	185.8
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-37}$	8	80	160.7
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-141}$	4	80	160.4
900	1024	3	7	$5e^{-11}$	$\leq 2^{-12}$	16	80	140.9
900	1024	4	7	$5e^{-11}$	$\leq 2^{-12}$	16	80	162.0
900	1024	3	7	$5e^{-11}$	$\leq 2^{-42}$	8	80	146.7
900	1024	3	7	$5e^{-11}$	$\leq 2^{-161}$	4	80	141.6
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-13}$	16	80	125.6
800	1024	4	7	$3.5e^{-10}$	$\leq 2^{-13}$	16	80	144.3
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-47}$	8	80	126.0
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-180}$	4	80	125.8
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-14}$	16	80	109.9
700	1024	12	2	$5.5e^{-09}$	$\leq 2^{-15}$	16	80	259.0
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-51}$	8	80	110.6
700	1024	4	6	$5.5e^{-09}$	$\leq 2^{-53}$	8	80	127.2
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-196}$	4	80	109.7
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-15}$	8	80	95.2
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-30}$	8	80	109.2
600	1024	13	2	$9.4e^{-08}$	$\leq 2^{-59}$	8	80	236.4
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-53}$	4	80	94.9
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-112}$	4	80	109.4

Table 4: Parameters and probability of error for FDFB

n	N	l	Bgbit	$\sigma_{\min}$	ϵ			$\lvert\mathcal{M}\rvert$	λ	time (s)
					worst case	Id	ReLU			
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-1}$	$\leq 2^{-11}$	$\leq 2^{-8}$	16	120	178.9
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-10}$	$\leq 2^{-62}$	$\leq 2^{-71}$	8	120	178.1
1024	1024	4	5	$7.8e^{-09}$	$\leq 2^{-33}$	$\leq 2^{-109}$	$\leq 2^{-115}$	8	120	202.3
1024	1024	5	5	$7.8e^{-09}$	$\leq 2^{-56}$	$\leq 2^{-125}$	$\leq 2^{-129}$	8	120	225.0
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-224}$	$\leq 2^{-326}$	$\leq 2^{-451}$	4	120	176.6
900	1024	5	4	$8.4e^{-08}$	≤ 0.72	$\leq 2^{-6}$	$\leq 2^{-4}$	16	120	198.6
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-2}$	$\leq 2^{-13}$	$\leq 2^{-17}$	8	120	177.7
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-5}$	$\leq 2^{-33}$	$\leq 2^{-40}$	8	120	198.2
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-44}$	$\leq 2^{-84}$	$\leq 2^{-196}$	4	120	177.1
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-115}$	$\leq 2^{-200}$	$\leq 2^{-367}$	4	120	247.4
800	1024	9	2	$5.9e^{-07}$	$\leq 2^{-1}$	$\leq 2^{-8}$	$\leq 2^{-10}$	8	120	252.4
800	1024	4	4	$5.9e^{-07}$	$\leq 2^{-4}$	$\leq 2^{-8}$	$\leq 2^{-21}$	4	120	157.5
800	1024	6	3	$5.9e^{-07}$	$\leq 2^{-12}$	$\leq 2^{-23}$	$\leq 2^{-62}$	4	120	196.2
800	1024	13	2	$5.9e^{-07}$	$\leq 2^{-21}$	$\leq 2^{-42}$	$\leq 2^{-109}$	4	120	322.4
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-7}$	$\leq 2^{-34}$	$\leq 2^{-32}$	16	80	176.1
1024	1024	4	7	$1.05e^{-11}$	$\leq 2^{-37}$	$\leq 2^{-37}$	$\leq 2^{-37}$	16	80	201.6
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-91}$	$\leq 2^{-135}$	$\leq 2^{-137}$	8	80	177.9
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-529}$	$\leq 2^{-544}$	$\leq 2^{-553}$	4	80	177.6
900	1024	3	7	$5e^{-11}$	$\leq 2^{-8}$	$\leq 2^{-39}$	$\leq 2^{-36}$	16	80	156.1
900	1024	4	7	$5e^{-11}$	$\leq 2^{-42}$	$\leq 2^{-42}$	$\leq 2^{-42}$	16	80	177.5
900	1024	3	7	$5e^{-11}$	$\leq 2^{-103}$	$\leq 2^{-154}$	$\leq 2^{-155}$	8	80	155.3
900	1024	3	7	$5e^{-11}$	$\leq 2^{-601}$	$\leq 2^{-618}$	$\leq 2^{-629}$	4	80	157.0
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-8}$	$\leq 2^{-43}$	$\leq 2^{-40}$	16	80	138.5
800	1024	4	7	$3.5e^{-10}$	$\leq 2^{-35}$	$\leq 2^{-47}$	$\leq 2^{-47}$	16	80	157.7
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-114}$	$\leq 2^{-172}$	$\leq 2^{-174}$	8	80	139.3
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-674}$	$\leq 2^{-694}$	$\leq 2^{-707}$	4	80	138.6
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-1}$	$\leq 2^{-24}$	$\leq 2^{-17}$	16	80	121.4
700	1024	12	2	$5.5e^{-09}$	$\leq 2^{-40}$	$\leq 2^{-53}$	$\leq 2^{-53}$	16	80	271.1
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-25}$	$\leq 2^{-123}$	$\leq 2^{-135}$	8	80	122.1
700	1024	4	6	$5.5e^{-09}$	$\leq 2^{-60}$	$\leq 2^{-170}$	$\leq 2^{-177}$	8	80	138.5
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-458}$	$\leq 2^{-595}$	$\leq 2^{-725}$	4	80	121.2
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-1}$	$\leq 2^{-6}$	$\leq 2^{-7}$	8	80	104.8
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-2}$	$\leq 2^{-16}$	$\leq 2^{-20}$	8	80	119.4
600	1024	13	2	$9.4e^{-08}$	$\leq 2^{-25}$	$\leq 2^{-133}$	$\leq 2^{-148}$	8	80	246.2
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-19}$	$\leq 2^{-37}$	$\leq 2^{-98}$	4	80	104.7
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-53}$	$\leq 2^{-103}$	$\leq 2^{-250}$	4	80	118.9

Table 5: Parameters and probability of error for the composition method

n	N	l	Bgbit	$\sigma_{\min}$	ϵ	$\|\mathcal{M}\|$	λ	time (s)	
								abs	ReLU
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-32}$	16	120	80.6	241.3
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-123}$	8	120	80.7	241.0
1024	1024	4	5	$7.8e^{-09}$	$\leq 2^{-137}$	8	120	92.1	277.4
1024	1024	5	5	$7.8e^{-09}$	$\leq 2^{-139}$	8	120	105.5	312.1
1024	1024	3	7	$7.8e^{-09}$	$\leq 2^{-482}$	4	120	80.6	240.9
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-29}$	16	120	91.8	274.7
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-66}$	8	120	81.4	247.7
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-109}$	8	120	91.9	275.1
900	1024	4	5	$8.4e^{-08}$	$\leq 2^{-255}$	4	120	81.5	243.6
900	1024	5	4	$8.4e^{-08}$	$\leq 2^{-427}$	4	120	94.8	276.9
800	1024	9	2	$5.9e^{-07}$	$\leq 2^{-48}$	8	120	120.1	358.9
800	1024	4	4	$5.9e^{-07}$	$\leq 2^{-30}$	4	120	72.2	216.5
800	1024	6	3	$5.9e^{-07}$	$\leq 2^{-89}$	4	120	91.5	273.9
800	1024	13	2	$5.9e^{-07}$	$\leq 2^{-154}$	4	120	157.1	469.7
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-36}$	16	80	80.0	239.3
1024	1024	4	7	$1.05e^{-11}$	$\leq 2^{-36}$	16	80	93.0	276.2
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-140}$	8	80	80.1	240.3
1024	1024	3	7	$1.05e^{-11}$	$\leq 2^{-554}$	4	80	80.2	241.3
900	1024	3	7	$5e^{-11}$	$\leq 2^{-40}$	16	80	70.6	210.8
900	1024	4	7	$5e^{-11}$	$\leq 2^{-41}$	16	80	81.1	242.8
900	1024	3	7	$5e^{-11}$	$\leq 2^{-159}$	8	80	70.8	211.0
900	1024	3	7	$5e^{-11}$	$\leq 2^{-630}$	4	80	70.8	212.3
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-45}$	16	80	62.8	188.1
800	1024	4	7	$3.5e^{-10}$	$\leq 2^{-45}$	16	80	72.2	216.3
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-179}$	8	80	63.4	188.9
800	1024	3	7	$3.5e^{-10}$	$\leq 2^{-708}$	4	80	63.4	189.7
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-49}$	16	80	54.8	165.9
700	1024	12	2	$5.5e^{-09}$	$\leq 2^{-52}$	16	80	130.1	388.4
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-191}$	8	80	55.4	165.9
700	1024	4	6	$5.5e^{-09}$	$\leq 2^{-201}$	8	80	63.4	190.0
700	1024	3	7	$5.5e^{-09}$	$\leq 2^{-753}$	4	80	54.9	164.3
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-37}$	8	80	47.8	142.6
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-85}$	8	80	54.9	163.8
600	1024	13	2	$9.4e^{-08}$	$\leq 2^{-220}$	8	80	119.0	354.5
600	1024	3	6	$9.4e^{-08}$	$\leq 2^{-139}$	4	80	47.7	142.0
600	1024	4	5	$9.4e^{-08}$	$\leq 2^{-331}$	4	80	54.9	163.8