Chimera: a unified framework for B/FV, TFHE and HEAAN fully homomorphic encryption and predictions for deep learning

Christina Boura[1,2], Nicolas Gama[1,3], and Mariya Georgieva[3,4,5]

[1] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, Versailles, France
[2] Inria, Paris, France
[3] Inpher, Lausanne, Switzerland
[4] Gemalto, Meudon, France
[5] EPFL, Lausanne, Switzerland

Abstract. This work describes a common framework for scale-invariant families of fully homomorphic schemes based on Ring-LWE, unifying the plaintext space and the noise representation. This new formalization allows to build bridges between B/FV, HEAAN and TFHE and provides the possibility to take advantage of the best of these three schemes. In particular, we review how different strategies developed for each of these schemes, such as bootstrapping, external product, integer arithmetic and Fourier series, can be combined to evaluate the principle nonlinear functions involved in convolutional neural networks. Finally, we show that neural networks are particularly robust against perturbations that could potentially result from the propagation of large homomorphic noise. This allows choosing smaller and more performant parameters sets.
Keywords: fully homomorphic encryption, Ring-LWE, neural network, floating point computation

1 Introduction

Homomorphic encryption provides the ability to operate on encrypted data without decrypting it. Soon after the development by Gentry of the first fully homomorphic encryption (FHE) scheme [19], many other schemes followed. In this paper we are interested in three scale-invariant HE-families: TFHE [14, 13] based on [20, 15], B/FV [3, 17, 8] and HEAAN [12, 11], that are nowadays among the most efficient constructions.

The three above schemes are all based on the Ring-LWE problem and their ciphertext space is the same up to rescaling by a factor. However, their plaintext spaces are all different, and as a consequence, each of the schemes has different properties and behaviour. For example, the B/FV message space allows to perform large vectorial arithmetic operations, as long as the multiplicative depth remains small. Generally speaking, B/FV-type operations necessary treat many slots in parallel, but are less efficient if each slot needs an individual treatment.

On the other side, TFHE performs very fast combinatorial operations on individual slots, as for example, automata, Boolean circuits and look-up table evaluations. Yet, this scheme is not designed for accomplishing repetitive tasks. TFHE usually tolerates large noise (so smaller parameters), and in the case of automata evaluation, the multiplicative depth is quite large. The third scheme, HEAAN, is a mixed encryption scheme dedicated to fixed point arithmetic and has shown to be very efficient for floating point computations. However, this scheme is not designed to produce exact integer computations.

Ideally, for use cases requiring homomorphic operations of different nature, one would like to take advantage of the best of these three worlds. For this reason, this work intends at defining a unified framework embodying these three schemes, and aims at introducing bridges that will permit to switch from one scheme to another. This unification becomes even more important with the recent standardization initiatives of homomorphic schemes and common APIs [5]. Moreover, the TFHE, B/FV (Seal) and HEAAN schemes are currently part of this standardization process.

This approach has many potential applications. In a scenario where operations on large datasets must be performed first and a decision must be taken on the result, the first part would be easy in the B/FV world, whereas the decision function can be easily evaluated in TFHE. Therefore, one would like to easily pass from B/FV to TFHE. If a lot of approximated computations have to be done, one would benefit from a connection from HEAAN to TFHE. In another similar example, many machine learning algorithms use SIMD operations to produce an output vector, and at the end one is interested in computing the maximum of its coordinates. Conversely, some banking systems need to perform many small computations on an encrypted database, with a possibly very large multiplicative depth during a long period of time, and at the end of the month provide statistics on the current dataset. In this case it is essential to operate in bootstrapped mode as in TFHE, and then do the low-depth statistical calculations in B/FV or HEAAN. In this case it is essential to have a bridge from transforming TFHE ciphertexts to B/FV or HEAAN ones.

Unifying the different approaches is not an easy task. Notably, as the plaintext spaces are all different, it is not obvious in some cases to find a meaningful semantic to the bridges. Another difficulty is the noise management. For example, HEAAN and B/FV have a radically different interpretation of noise. In B/FV, computations on the plaintext are discrete and exact. Instead, HEAAN follows the floating point model with fixed precision to represent a continuous plaintext space. Each operation adds a new error and mathematically this can be expressed as a loss of one bit of precision after each operation. Yet, fixed point arithmetic is commonly used in long chains of computations: it is the numerical stability of the global algorithm that corrects the result. In the introduction of [11], the authors cite notably the examples of control systems with negative feedback, or of gradient descent. In this paper, we extend these results by measuring the effect of various perturbations models that correspond to FHE approximation errors, on small, medium and large neural networks.

The theory of TFHE unites these discrete/exact vs. continuous/approximate visions and allows to interconnect all the worlds. By unifying the schemes, we obtain some interesting observations. For example, the relinearisation in the internal products of HEAAN and B/FV can now be expressed in terms of the external TFHE product, and without loss as it is the same algorithm. Another example is the functional keyswitch of TFHE, which can be used as a flexible technique to manipulate, pack and unpack values, or apply linear maps to B/FV slots. It is a valid alternative to the multi-Hadamard product, especially when switching from a small key to a large one.

For a given nonlinear function (e.g. absolute value, rounding, comparison, division), and depending on the desired level of precision in the output, we have different options for evaluating it (choosing TFHE, B/FV or HEAAN). With the bridges proposed in this paper, we can easily alternate between the representations and between the arithmetic and logic/combinatorial phases.

Our Contributions

Unified framework and bridges. Our first contribution is a unified representation of three of the most promising homomorphic schemes: B/FV, TFHE and HEAAN. In order to exploit the advantages of each scheme, we need to be able to homomorphically switch between the different plaintext spaces. We propose to reexpress all the plaintext spaces as subsets of the same module, and to use the real distance on the torus to quantify the transformation errors. After that, we introduce all the necessary bridges to interconnect the B/FV, TFHE and HEAAN schemes as described in Figure 1. Moreover, with this unified representation the analytic techniques described in TFHE [13] can be used to analyze the average noise propagation in the three schemes.

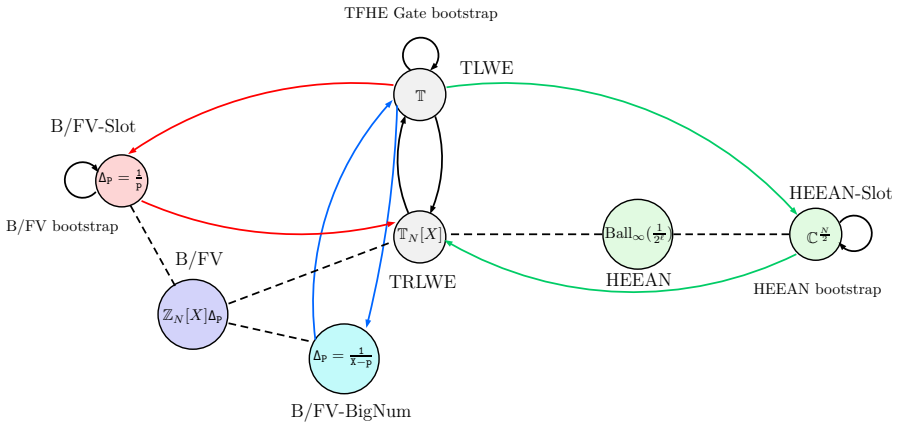

Fig. 1: Bridges between R-LWE homomorphic schemes.

Modeling of homomorphic operations and of the noise for convolutional neural networks followed by experiments. In the second part we show how one can homomorphically evaluate with the above schemes most of the deep learning crucial functions (e.g. absolute value, sign, max, `ReLU`) and discuss about the potential difficulties of such an approach. We model the noise generated during the homomorphic operations and the loss of precision by adding a Gaussian error to the output of each non-linear function. This model permits us to simulate homomorphic evaluations with large amounts of noise, and predict the effect on the classification accuracy, without the need of evaluating heavy and time-consuming homomorphic operations. This allows to determine the best CNN structure and the smallest FHE parameters required during a pre-study phase.

We performed experiments with perturbations on three different convolutional neural networks of small (LeNet-5), medium (cat-and-dog-9) and large (ResNet-34) size and we observed that all the neural network we tested support quite large relative errors of at least 10%, without almost any impact on the global accuracy. It means that only 4 bits of precision (instead of 20 to 40 bits usually) are needed on all fixed point operations throughout the network, which results in very small parameter sets and fast homomorphic operations.

2 Background

In this paper we denote by $\mathbb{T}$ the real Torus $\mathbb{R}/\mathbb{Z}$, the set of real numbers modulo 1. We denote by $\mathbb{Z}_N[X] = \mathbb{Z}[X]/(X^N+1)$ the ring of polynomials with integer coefficients module X^N+1.

Respectively,$\mathbb{R}_N[X] = \mathbb{R}[X]/(X^N+1)$ is the ring of real polynomials module X^N+1.

We denote by $\mathbb{T}_N[X] = \mathbb{R}_N[X]/\mathbb{Z}_N[X]$ (a.k.a $\mathbb{R}[X] \mod X^N+1 \mod 1$). Note that $\mathbb{T}_N[X]$ is a $\mathbb{Z}_N[X]$-module and not a ring. It does not have an internal multiplication, but it has an external multiplication with coefficients in $\mathbb{Z}_N[X]$.

We denote also $\mathbb{B}_N[X]$ the subset of $\mathbb{Z}_N[X]$ with binary coefficients.

We start here by a brief description of the three scale-invariant families of HE schemes treated in this work. All of them are based on the Ring-LWE problem.

B/FV (Brakerski, Fan-Vercauteren, and Seal): In this scheme, the message space is the ring $\mathcal{R}_p = \mathbb{Z}[X]/(X^N+1) \bmod p$, which has an internal addition and multiplication. A message $\mu \in \mathcal{R}_p$ is encrypted on a larger modulus ring, as $(a,b) \in \mathcal{R}_q^2$, where $\mathcal{R}_q = \mathbb{Z}[X]/(X^N+1) \bmod q$, a chosen is uniformly at random in $\mathcal{R}_q$ and b is close to $s.a + \lceil \frac{q}{p}\mu \rfloor$, up to some small Gaussian error. Homomorphic addition of two ciphertexts (a_1,b_1) with (a_2,b_2) is performed termwise on the two components, and homomorphic multiplication consists in expanding $\mu_1\mu_2 = \mu$ as the polynomial equation $\frac{p}{q}(b_1+a_1.s)\frac{p}{q}(b_2+a_2.s) = \frac{p}{q}(b+a.s)$, and identifying its coefficients a and b. Since the modulo-p and modulo-q classes are not compatible, all these products are actually performed on small representatives in the real ring $\mathbb{R}[X]$. Also, the quadratic term $\frac{p}{q}a_1a_2.s^2$ in the left hand side is relinearized: it is bit-decomposed as a small public integer combination

of the secret vector $(s^2, 2s^2, 4s^2, ..., \frac{q}{2}s^2)$ and reencrypted as the same combination of $\mathsf{KS} = (\mathrm{Enc}(s^2), \mathrm{Enc}(2s^2), ..., \mathrm{Enc}(\frac{q}{2}s^2))$ where KS is a public precomputed vector of ciphertexts called "KeySwitching key". More details and formal definitions are given in the original papers (see [3, 17]). The noise amplitude grows by a small factor $O(N)$ on average after each multiplication, so it is a common practice to perform a *modulus-rescaling* step, that divides and rounds each coefficient as well as the modulus q by an equivalent amount, in order to bring the noise amplitude back to $O(1)$, so that the subsequent operations continue on smaller ciphertexts.

HEAAN In this scheme, the message space is the set of polynomials of $\mathcal{R}_q$ of small norm $\leq B$, with some uncertainty on the least significant bits, which means that the $O(1)$ least significant bits of the number are considered as noise and only the most significant bits of μ are required to have an exact decryption. A HEAAN ciphertext is also a Ring-LWE tuple $(a, b) \in \mathcal{R}_q^2$ where a is uniformly random in $\mathcal{R}_q$, and b is close to $a.s + \mu$, i.e. up to a Gaussian error of amplitude $O(1)$. This time, plaintexts and ciphertexts share the same space (no $\frac{p}{q}$ rescaling). Multiplication of two messages uses the same formula as in B/FV, including relinearization with a keyswitch: if both input messages are bounded by B with $O(1)$ noise, the product is a message bounded by B^2 with noise $O(B)$. At this point, it is a common practice to perform a modulus-rescaling step that divides everything by B to bring the noise back to a $O(1)$ level. Unlike B/FV, this division in the modulus switching scales not only the ciphertext, but also the plaintext by B. This can be fixed by adding a (public) tag to the ciphertext to track how many of these divisions by B have been performed.

TFHE Here, messages and ciphertexts are expressed over the torus modulo 1 ($\mathbb{T} = \mathbb{R}/\mathbb{Z}$). Each operation (on plaintexts or ciphertexts) is done at a given noise amplitude $\alpha \ll 1$. This means that only $\log_2(\alpha)$ fractional bits of precision are considered, and least-significant bits that are left behind become part of the ciphertext's noise. TFHE can represent three plaintext spaces, and various morphisms, or actions between them.

- TLWE encodes individual (continuous) messages over the torus $\mathbb{T}$;
- TRLWE encodes (continuous) messages over $\mathbb{R}[X] \mod (X^N + 1) \mod 1$, which can be viewed as the packing of N individual coefficients;
- TRGSW encodes integer polynomials in $\mathbb{Z}_N[X]$ with bounded norm.

In TFHE, TLWE (resp. TRLWE) ciphertexts of a message μ have the same shape $(a, b = s.a + \mu + e)$ where a is uniformly random in $\mathbb{T}^N$ (resp. $\mathbb{T}_N[X]$) and e is a small zero-centered error.

TLWE (resp. TRLWE) is built around a secret Lipschitzian functional called *phase* parametrized by a small (in general binary) secret key $\mathbf{s}$, and usually defined as $\varphi_s : \mathbb{T}^N \to \mathbb{T}$ (resp. $\varphi_s : \mathbb{T}_N[X]^2 \to \mathbb{T}_N[X]$) that maps $(\mathfrak{a}, \mathfrak{b})$ to $\mathfrak{b} - s.\mathfrak{a}$ (resp. $b(X) - s(X).a(X)$). The fact that the phase is a Lipschitz function

makes the decryption tolerant to errors and allows working with approximated numbers.

TFHE proposes an analytic description of homomorphic operations where each ciphertext is assimilated to a random variable that depends on the randomness of a and e. In this case, the message and the error amplitude can be analytically written as the *expectation* and *standard deviation* of the phase $\varphi_s(a,b) = b - sa = \mu + e$ of the ciphertext. The expectation, variance and standard deviation on the torus are well defined only for concentrated distributions whose support is included in a ball of radius $\frac{1}{4}$ (up to negligible tails). This is the case of the error distribution of T(R)LWE. More information on the definition of expectation and standard deviation for concentrated distributions on the torus can be found in [14]. The benefit of the definition of the message via the expectation of the phase is that it is valid with infinite precision on any (possibly continuous) subset of $\mathbb{T}_N[X]$. However, this definition is only analytic in the sense that it can be used to prove the correctness of a cryptosystem by induction, but it is not a definition that can be used for performing computations, since in practice the expectation of the phase cannot be computed from a single sample of the distribution.

We describe below the algorithms that are used for the TFHE with TRLWE encryption scheme.

Parameters: A security parameter λ, and a minimal noise α (implicitly defining a minimal key size N).

KeyGen/Phase: A uniformly random binary key $s \in \mathbb{B}_N[X]$, this implicitly defines the secret *phase* function $\varphi : \mathbb{T}_N[X]^2 \to \mathbb{T}_N[X], (a,b) \mapsto (b - sa)$.

Encrypt $(\mu, \mathbf{s}, \alpha)$: Pick a uniformly random $a \in \mathbb{T}_N[X]$, and a small Gaussian error $e \leftarrow \mathcal{D}_{\mathbb{T}_N[X],\alpha}$ and return $(a, s.a + \mu + e)$.

DecryptApprox(c, s): (approx) Return $\varphi_s(c)$, which is close to the actual message.

Decrypt$(c, s, \mathcal{M})$: (rounded) Round $\varphi_s(c)$ to the nearest point in $\mathcal{M}$.

Message(c, s): (analytic) The message is the expectation of $\varphi_s(c)$ (across all possible randomizations of noise).

Public linear combination over $\mathbb{Z}_N[X]$: return $\sum e_i.c_i$.

External product Given a TRGSW ciphertext A of $M \in \mathbb{Z}_N[X]$ and a TRLWE ciphertext b of $\mu \in \mathbb{T}_N[X]$, compute $A \boxdot_\alpha b$ at precision $\alpha > 0$, which encrypts $M.\mu \in \mathbb{T}_N[X]$ (see [13]) $\boxdot_\alpha : \mathsf{TRGSW} \times \mathsf{TRLWE} \longrightarrow \mathsf{TRLWE}$

Apply a $\mathbb{Z}$-module morphism f: use a Keyswitch algorithm (Theorem 2).

To ease the reading of this paper, we specify only one particular instantiation of TFHE, as described in [13] where $k = 1$ and $\beta = 1$ (all bit-decompositions are binary). We present all theorems with decomposition in base 2 (for bootstrapping, keyswitch, external product and bitdecomp) to minimize the number of parameters in the formulas.[6].

[6] With this choice, the parameters from [13] correspond to: $k = 1$, $\beta = 1$, $Bg = 2$, $\ell = -\log(\alpha)$, $t = \ell$, $n = N$, $VKS = VBK = \alpha^2$, which implies $\epsilon = \alpha/2$. As usual, a non binary decomposition is possible and gives small poly-logarithmic improvements.

To perform an operation at precision $\alpha \ll 1$ (i.e. noise standard deviation α during the operation), we always use $\ell = -\log_2(\alpha)$ terms in every bit decomposition.

The primary definition of α is the noise's standard deviation during the current operation: α is thus not a static parameter of the framework (noise rate, precision and approximate decomposition error, keyswitch noise and bootstrapping noises are thus equal or strongly related).

Any TLWE, TRLWE, TRGSW ciphertext, bootstrapping key or keyswitching key given at a higher precision, can always be rounded and truncated to match the current precision α. Working with precision α implies a minimal size for the current binary key (roughly $N \approx \max(256, 32\alpha)$ from the security estimates for a security parameter of $\lambda = 128$ bits (see Section 6 of [13]).

Whenever α varies (e.g. increases after each multiplication, or decreases after a bootstrapping), we always use the last keyswitching and bootstrapping operation to switch to a new encryption key whose entropy is as close as possible to the lower bound $N \approx \max(256, 32\alpha)$ from the security estimates.

External Product In this case, we have for the external product the following theorem (see [13]):

Theorem 1. *(External Product) Let A be* TRGSW *ciphertext of message $\mu_A \in \mathbb{Z}_N[X]$ and b be an (independent)* TRLWE *ciphertext of $\mu_b \in \mathbb{T}_N[X]$. Then, there exists a homomorphic external product algorithm, noted $A \boxdot_\alpha b$ (first introduced in [4] and formalized on the torus in [14]), such that $\mathsf{Message}(A \boxdot_\alpha b) = \mu_A \cdot \mu_b$ and $\mathsf{Var}(\mathsf{Err}(A \boxdot_\alpha b)) \leq 2\ell N \mathsf{Var}(\mathsf{Err}(A)) + \frac{1+N}{4} \|\mu_A\|_2^2 \alpha^2 + \|\mu_A\|_2^2 \mathsf{Var}(\mathsf{Err}(b))$.*

This theorem will in general be used to multiply a TRLWE ciphertext b with a precomputed TRGSW ciphertext (e.g. a ciphertext of the binary secret key in the case of bootstrapping): in this case, we can choose $\mathsf{Var}(\mathsf{Err}(A)) = \alpha^2$, and we have $\|\mu_A\|_2^2 \leq N$ (or even $\|\mu_A\|_2^2 = 1$ if TRGSW ciphertexts encrypt only single bits, as in [14] and [13]). In this case, the working precision α equals the targeted output precision divided by N, so that the first two error terms in the theorem remain negligible.

KeySwitching. In order to switch between the scalar and polynomial message space $\mathbb{T}$ and $\mathbb{T}_N[X]$, the authors of [13] generalized the notions of sample extraction and keyswitching. On the one side a $\mathsf{PubKS}(f, \mathsf{KS}, \mathsf{c}_1, \ldots, \mathsf{c}_p)$ algorithm homomorphically evaluates linear morphisms f from any $\mathbb{Z}$-module $\mathbb{T}^p$ to $\mathbb{T}_N[X]$ using the functional keyswitching keys KS. It is possible to evaluate also a private linear morphism, but the algorithm is slower.

On the other side, the algorithm $\mathsf{SampleExtract}_i(\mathbf{c})$ allows to extract from $\mathbf{c} = (\mathbf{a}, b) \in \mathsf{TRLWE}_K(\mu)$ the TLWE sample that encodes the i-th coefficient μ_i with at most the same noise variance or amplitude as $\mathbf{c}$.

Theorem 2. *(Functional Key Switch) Given p* TLWE *ciphertexts $\mathsf{c}_i \in \mathsf{TLWE}_S(\mu_i)$, a public R-lipschitzian morphism $f : \mathbb{T}^p \to \mathbb{T}_N[X]$ of $\mathbb{Z}$-modules, and $\mathsf{KS}_{i,j} \in \mathsf{T(R)LWE}_{K,\alpha}(\frac{S_i}{2^j})$ with standard deviation α and S_i the i-th coefficient of the key*

S, Algorithm 2 described in [13] outputs a $\mathsf{T(R)LWE}$ *sample* $\mathbf{c} \in \mathsf{T(R)LWE}_K(f(\mu_1, \ldots, \mu_p))$ *such that:* $\mathsf{Var}(\mathsf{Err}(\mathbf{c})) \leq R^2 \max(\mathsf{Var}(\mathsf{Err}(\mathfrak{c}_i))) + \alpha^2(\ell N^2 + \frac{N}{4})$.

Gate bootstrapping. The gate bootstrapping in TFHE is a method to refresh noisy TLWE ciphertext $\mathfrak{c}$, and it can also change its plaintext space. For higher level point of view the gate bootstrapping algorithm allows to evaluate any pointwise defined negacyclic function $f : \mathbb{T} \to \mathbb{T}$ to the phase of a TLWE sample.

Theorem 3. *(Functional Bootstrap) Given a* TLWE *ciphertext* $\mathfrak{c}$ *on* n *bits key* K*, which has been rounded to* $(\frac{1}{2N}\mathbb{Z}/\mathbb{Z})^2$*, a negacyclic function* $f : \frac{1}{2N}\mathbb{Z}/\mathbb{Z} \to \mathbb{T}$ *defined pointwise* $[f(\frac{0}{2N}), \ldots, f(\frac{N-1}{2N})]$ *and a bootstrapping key* $BK = \mathsf{TRGSW}_S(K_i)$ *with standard deviation* α*, Algorithm 10 described in [13] outputs a* TLWE *sample* $\mathfrak{c}' \in \mathbb{T}$ *encrypted* $f(\varphi_K(\mathfrak{c}))$ *with key* S *such that:* $\mathsf{Var}(\mathsf{Err}(\mathfrak{c}')) \leq \alpha^2 n(2\ell N + N + \frac{5}{4} + \ell)$.

Until now, the most frequent non-linear function used in Bootstrapping is the rounding function, since rounding the phase of a ciphertext is equivalent to decrypting it, and homomorphic decryption is the noise-reduction method proposed by Gentry in 2009 [19].

3 Unified framework for scale invariant homomorphic schemes

3.1 Unifying all message spaces

Interestingly, the three homomorphic schemes use nearly the same ciphertext space (up to rescaling by a factor q), and the notion of phase can be ported to all three up to minor differences ($b - sa$ versus $b + sa$ in B/FV, or $\sum a_i s^i$ in Seal). However, plaintext spaces are completely different as they are based on different mathematical structures (groups, rings, intervals, random sets). In order to exploit the advantages of each scheme, we need to be able to homomorphically switch between the different plaintext spaces, and most importantly, to give a meaningful semantic to these transformations. Here, we propose to reexpress all the plaintext spaces as subsets of the same module $\mathbb{T}_N[X]$, and to use the real distance on the torus to quantify the transformation error.

In this setting, all schemes use the same ciphertext space $\mathbb{T}_N[X]^2$, the same key space $\mathbb{B}_N[X]$, and the same phase function $\varphi_s(a, b) = b - s.a$. Thus, the analytic characterization of the message and error as respectively the expectation and the standard deviation of the phase from TFHE is automatically ported to all schemes. TFHE has two interpretations of the concrete decryption: the first one says that the phase is always close (within a distance α) to the actual message, and is a good enough approximation thereof. This is the default approach in the HEAAN cryptosystem on approximate numbers, where only the significant digits matter, and accumulated errors are not corrected by the cryptosystem (but rather by the numerical stability of the physical system that is homomorphically evaluated). The second approach consists in restricting the valid message space

to a discrete subset of $\mathbb{T}_N[X]$ with a packing radius $\geq \alpha$. In this case, which corresponds to the B/FV setting, the exact message can be recovered by rounding the phase to the closest valid message.

Preserving the user slots. Even if we unify the native plaintext of all schemes on $\mathbb{T}_N[X]$, we would like to preserve the notion of user-side slots, which corresponds to the way the end-user actually employs the schemes.

In B/FV, homomorphic operations are presented to the user either as N SIMD slots modulo a medium-sized integer p (in [17, 8]), or more recently as a single big-number (or rational) slot modulo p^N+1 [9]. Exact operations on parallel slots can be used for example in financial applications, and exact big-number operations can be applied to the homomorphic evaluation of cryptographic functions like RSA or DSA. In all cases, these slots are isomorphic to a principal ideal of $\mathbb{Z}_N[X]$, which is viewed as the *native plaintext* in [9]. In our formalization, we preserve the user-space slots, and we reexpress the native plaintext space as a principal ideal sublattice $\mathcal{M}$ of $\mathbb{T}_N[X]$. Rounding the phase of a ciphertext to the native plaintext space means solving a bounded distance decoding (BDD) problem on the lattice $\mathcal{M}$, which is easy if we know a very small generator of $\mathcal{M}$ (which is the case in [8, 17, 9]). We formalize this in Section 3.2.

In HEAAN, homomorphic operations are presented as $N/2$ SIMD slots containing fixed-point complex numbers, with the same public exponent and the same precision. By interpolation on the complex roots of $X^N + 1$, the native plaintext can be mapped to small polynomials of $\mathbb{T}_N[X]$ (e.g. a ball of small fixed radius). We formalize this in Section 3.3.

Finally, in TFHE, the message space is an arbitrary subset of $\mathbb{T}_N[X]$: the only constraint is the local stability of each operation. For example in the gate bootstrapping, fresh/bootstrapped LWE samples encode Boolean values on the plaintext space $\{-\frac{1}{8}, +\frac{1}{8}\}$ which is not a group. The computation of binary gates uses exclusively a linear combination of two fresh samples with coefficients $\in \{-1, 0, 1\}$ (which temporarily brings the message space to $\{-\frac{1}{4}, 0, +\frac{1}{4}\}$), followed by a bootstrapping. In this case, the packing radius is $\frac{1}{8}$, which is larger than the $\frac{1}{16}$ we would get if TFHE was described on the smallest group that contains these plaintexts.

The unified native plaintext spaces and the correspondance with the user-space slots are shown in Figure 2.

3.2 A general abstraction of B/FV over the torus

B/FV uses a ring space, which means that the plaintext space has an internal multiplication. In order to support approximations, the multiplication is lifted to multiplication over the real ring $\mathbb{R}_N[X]$. Depending on the choice of the plaintext ring, usually presented as a subring of $\mathbb{Z}_N[X]$, and up to ring isomorphism, this allows the user to work on more friendly slots, such as a vector of independent integers in $\mathbb{Z}/p\mathbb{Z}$ for a medium-sized integer p (see [17, 3]), a vector of elements in a finite field (see [24]), or recently, a big integer or rationals modulo $p^N + 1$ (see [9]) via the NTRU trick (see [26]). The B/FV scheme is very efficient in

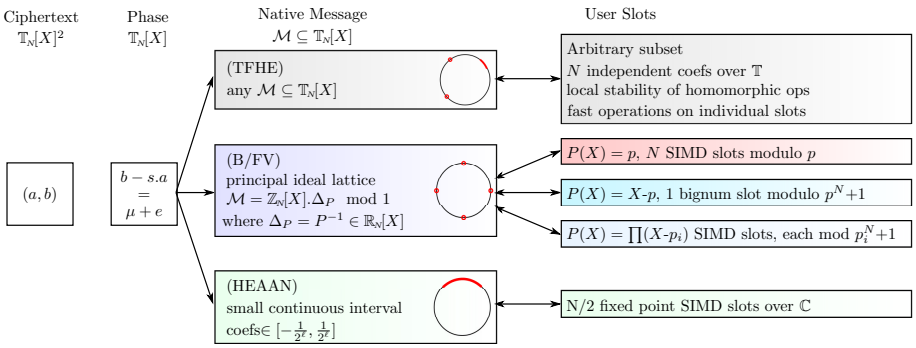

Fig. 2: Unifying the plaintext space in RLWE-schemes

evaluating arithmetic circuits (i.e. polynomials within the native plaintext ring structure), however huge slowdowns are noticed when evaluating comparisons, the sign function, or other non-linear decision diagrams that do not correspond to sparse low depth polynomials. Here, we propose an alternative that consists in switching to a different scheme, and for instance, execute the non-arithmetic operation via TFHE's gate bootstrapping. We explain how to unify the plaintext spaces to enable this conversion.

For this, we adopt the dual description that is used to tackle both independent slots and the big-numbers case in [18, 9]. Given an integer polynomial $P(X) \in \mathbb{Z}[X]$ relatively prime to X^N+1, we can identify $P(X)$ with its negacyclic matrix $\mathtt{P}$ (containing its coefficients). Let $\mathtt{\Delta_P} = \mathtt{P}^{-1}$ be the matrix of the inverse of $P(X)$ in $\mathbb{R}_N[X]$. We call respectively $\mathcal{L}(\mathtt{P})$ and $\mathcal{L}(\mathtt{\Delta_P})$ the real lattices generated by the rows of $\mathtt{P}$ and $\mathtt{\Delta_P}$. These lattice are dual to each other.

The native plaintext space $\mathcal{M}$ is the submodule of $\mathbb{T}_N[X]$ generated by $\Delta_P(X)$. Geometrically, it corresponds to the vectors in the lattice $\mathcal{L}(\mathtt{\Delta_P})$ whose coefficients are reduced modulo 1. By duality, $\mathcal{M}$ is also the set of all the real polynomials $\mu(X) \in \mathbb{R}_N[X]$ such that μP is integer (mod $X^N + 1$).

Algebraically, $\mathcal{M}$ is in bijection with the integer ring defined as $\mathfrak{R}_P = \mathbb{Z}[X] \mod X^N + 1 \mod P(X)$ and this is the most frequently used representation in the literature. We verify that the two plaintext spaces are isomorphic and that multiplication by $\Delta_P(X)$ is the isomorphism.

Lemma 1. *The right multiplication by $\Delta_P(X)$ is a $\mathbb{Z}_N[X]$-module isomorphism from $\mathcal{R}_P = \mathbb{Z}[X] \mod X^N + 1 \mod P(X)$ to $\mathcal{M}$.*

Proof. Consider the mapping $\psi : \mathbb{Z}_N[X] \to \mathcal{M}, U(X) \mapsto U.\Delta_P$. This application is surjective by definition. We now compute $\ker(\psi)$. Let $U \in \mathbb{Z}_N[X]$ such that $U(X).\Delta_P(X) = 0 \in \mathbb{T}_N[X]$.
By definition, there exists an integer polynomial $V(X) \in \mathbb{Z}_N[X]$ such that $U(X).\Delta_P(X) = V(X) \in \mathbb{R}_N[X]$. Multiplying both sides by $P(X)$, we get $U(X) = V(X)P(X)$, which is an integer multiple of P. Reciprocally, all integer multiples of P are sent to 0 by ψ, so $\ker(\psi) = \mathbb{Z}_N[X].P$. By the isomorphism

theorem, we can conclude that multiplication by Δ_P is an isomorphism from $\mathbb{Z}_N[X] \mod P$ to $\mathcal{M}$, and thererefore provides an induced ring structure on $\mathcal{M}$. □

By the isomorphism, the native plaintext space $\mathcal{M}$ has an induced internal multiplication, which can be written as the following Mongomery product:

Definition 1 (Native plaintext product).

$$\begin{aligned} &\mathcal{M} \times \mathcal{M} \to \mathcal{M} \\ &(\mu_1, \mu_2) \mapsto \mu_1 \boxtimes_P \mu_2 = (P \times \mu_1 \times \mu_2) \mod X^N + 1 \mod 1. \end{aligned} \tag{1}$$

Here, the product $P \times \mu_1 \times \mu_2$ is done over $\mathbb{R}[X]$ by choosing any real representative for μ_1, μ_2.

We now explain how this product handles approximations of μ_1 and μ_2 with small continuous errors, if used for TRLWE ciphertexts.

Multiplication in B/FV We now review how the exact multiplication over plaintexts extends to an approximate B/FV multiplication over TRLWE samples. Given a TRLWE ciphertext $c \in \mathbb{T}_N[X]^2$, we call $(a, b) \in \mathbb{R}_N[X]^2$ the smallest representative of c if $c = (a, b) \mod 1$ and a, b have all their coefficients in $[-\frac{1}{2}; \frac{1}{2})$.

Let $s \in \mathbb{B}_N[X]$ be a TRLWE key, and RK be a relinearization key, which is a TRGSW encryption of s with key s. Let c_1, c_2 be two TRLWE ciphertexts under the same key s. We note $c_1 \boxtimes_{P,\alpha} c_2$ the following product:

Definition 2 (Internal homomorphic product). *Let $\mathsf{RK} = \mathsf{TRGSW}_s(s)$ be a relinearization key, i.e. a self-encryption of a key $s \in \mathbb{B}_N[X]$ with noise standard deviation $\alpha > 0$, and let P be an integer polynomial. Let $c_1 = (a_1, b_1)$, $c_2 = (a_2, b_2)$ be two* TRLWE *ciphertexts. We denote by (a'_i, b'_i), (a'_2, b'_2) the smallest representatives of the ciphertexts over $\mathbb{R}_N[X]$ with coefficients in $(-\frac{1}{2}, \frac{1}{2})$ and $C_0 = P \times b'_1 \times b'_2$, $C_1 = P \times (a'_1 \times b'_2 + a'_2 \times b'_1)$ and $C_2 = P \times a'_1 \times a'_2$. We define*

$$c_1 \boxtimes_{P,\alpha} c_2 = (C_1, C_0) - RK \boxdot_\alpha (C_2, 0). \tag{2}$$

Note that this definition of the internal B/FV product relies on a precomputed TRGSW external product (which is faster). This connection between the two products becomes possible because the plaintext space of B/FV and TFHE is unified. A formulation closer to the original relinearization presented in B/FV would rather take RK' an encryption of s^2, and compute $(C_1, C_0) + RK' \boxdot_\alpha (0, C_2)$, but this approach generates more noise in the case of a partial bit-decomposition since $\left\|s^2\right\|_2 \geq \|s\|_2$.

Lemma 2 (B/FV noise propagation). *Let RK, c_1, c_2 be a relinearization key and two* TRLWE *ciphertexts of $\mu_1, \mu_2 \in \mathcal{M} = \mathbb{Z}_N[X].\Delta_P$, with the same key $s \in \mathbb{B}_N[X]$ as in* Definition *2. Then the message of $c_1 \boxtimes_{P,\alpha} c_2$ is the product $\mu_1 \boxtimes_P \mu_2$ of the two individual messages, and the noise variance satisfies*

$$\mathsf{Var}(\mathsf{Err}(c_1 \boxtimes_{P,\alpha} c_2)) \leq \frac{1+N+N^2}{2} \|P\|_2^2 \max(\mathsf{Var}(\mathsf{Err}(c_i)) + \left(2\ell N + \frac{N^2+N}{4}\right) \alpha^2 \tag{3}$$

Proof. Let $\mu_1, \mu_2, e_1, e_2 \in \mathbb{R}_N[X]$ be the smallest representatives of respectively the message and error of c_1 and c_2. By definition, for each $i = 1, 2$ we have $b_i - s.a_i = \mu_i + e_i + I_i$ where I_i is an integer and the variance of I_i is $\leq N$.

$$\begin{aligned}
\varphi_s(c_1 \boxtimes_{P,\alpha} c_2) &= C_0 + s.C_1 + s^2.C_2 + \mathsf{Err}(RK \boxdot_\alpha (0, C_2)) \mod 1 \\
&= (b_1 - sa_1)(b_2 - sa_2)P + \mathsf{Err}(RK \boxdot_\alpha (0, C_2)) \mod 1 \\
&= (\mu_1 + e_1 + I_1)(\mu_2 + e_2 + I_2)P + \mathsf{Err}(RK \boxdot_\alpha (C_2, 0)) \mod 1 \\
&= \mu_1\mu_2 P + e_1\mu_2 P + e_2\mu_1 P + e_1e_2 P + e_1I_2 P + e_2I_1 P \\
&+ \mathsf{Err}(RK \boxdot_\alpha (C_2, 0)) \mod 1 \\
&= \mu_1 \boxtimes_P \mu_2 + e_1\mu_2 P + e_2\mu_1 P + e_1e_2 P + e_1I_2 P + e_2I_1 P \\
&+ \mathsf{Err}(RK \boxdot_\alpha (C_2, 0)) \mod 1
\end{aligned}$$

Taking the expectation, since all multiples of e_i as well as $\mathsf{Err}(RK \boxdot (0, C_2))$ have a null expectation, the message of $c_1 \boxtimes_{P,\alpha} c_2$ is $\mu_1 \boxtimes_P \mu_2$. By bounding the variance of each error term, we prove Eq. (3). □

The working precision α should be chosen approximatively equal to the standard deviation of the input ciphertexts, so that the term in α^2 remains negligible compared to the first one in (3). Thus, the noise standard deviation multiplicative overhead is bounded by $O(N||P||_2)$ in the average case.

Bridging B/FV slots with TFHE. In the classical description of B/FV or Seal, $P(X)$ is usually chosen as a constant integer p. In this case, the plaintext space $\mathcal{M}$ consists in all the multiples of $\Delta_P = \frac{1}{p} \mod (X^N + 1) \mod 1$, which is a rescaling of the classical plaintext space description $\mathbb{Z}/p\mathbb{Z}[X]/(X^N + 1)$. In particular, if $X^N + 1$ has N roots modulo p, $\mathcal{M}$ is isomorphic to N independent integer slots modulo p (else, there are less slots, in extension rings or fields). From a lattice point of view, $\mathcal{M}$ is assimilated to the orthogonal lattice generated by $\frac{1}{p}I_N$ (I_N is an integer). The packing radius of $\mathcal{M}$ is $\frac{1}{2p}$, which is the maximal error that can be tolerated on the phase. Rounding an element to $\mathcal{M}$ consists in rounding each coordinate independently to the nearest multiple of $\frac{1}{p}$.

In the literature, the isomorphism used to obtain the slot representation is

$$\begin{aligned}
\mathbb{Z}[X] \mod (X^N + 1) \mod p &\to (\mathbb{Z}/p\mathbb{Z})^N \\
\mu &\mapsto (\mu(r_1), \ldots, \mu(r_N)),
\end{aligned}$$

where $r_1, \ldots, r_N$ are the N roots of the polynomial $(X^N + 1) \mod p$.

This isomorphism allows to manipulate N independent slots in parallel. Typical values are $N = 2^{15}$ and $p = 2^{16} + 1$ (allowing a very small noise $\alpha \approx 2^{-886}$ according to [5], so a multiplicative depth of ≈ 28 without keyswitch according to the propagation of Lemma 2).

If we identify a polynomial with its coefficient vector in $(\mathbb{Z}/p\mathbb{Z})^N$, the isomorphism between the coefficients and the slot corresponds to the Vandermonde

matrix of $(r_0, \ldots, r_{N-1}) \mod p$

$$\mathsf{VDM} = \begin{bmatrix} 1 & r_1^1 & \cdots & r_1^{N-1} \\ 1 & r_2^1 & \cdots & r_2^{N-1} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & r_N^1 & \cdots & r_N^{N-1} \end{bmatrix} \mod p. \tag{4}$$

B/FV → TFHE. Suppose that we need to extract the slots of a B/FV ciphertext in order to process them with TFHE. From the slots $(z_1, \ldots, z_N)$, we would need to obtain a polynomial whose coefficients are $\frac{1}{p}(z_1, \ldots, z_N)$. More generally, suppose that we want to evaluate a $\mathbb{Z}$-module morphism f from $(\mathbb{Z}/p\mathbb{Z})^N \to (\mathbb{Z}/p\mathbb{Z})^N$ of a matrix F and get the output as the polynomial $\sum_{i=0}^{N-1} \mu'_i X^i$ where $(\mu'_0, \ldots, \mu'_{N-1}) = \frac{1}{p} f(z_1, \ldots, z_N)$.

Then, by definition, we have $\mu' = (F.\mathsf{VDM})\mu \mod 1$, where $F.\mathsf{VDM}$ can be any integer representative of $F.\mathsf{VDM} \mod p$. In particular, we can always take the representative with all coefficients in $[-\frac{p}{2}, \frac{p}{2}]$. This is a $\frac{Np}{2}$-lipschitzian $\mathbb{Z}$-module morphism, and can be evaluated via the functional keyswitch of Theorem 2. Coefficients can then be homomorphically extracted as individual TLWE ciphertexts with the SampleExtract of TFHE.

Proposition 1 (B/FV slots → TFHE). *Let $c = (a, b)$ be a TRLWE ciphertext encoding N slots $(z_1, \ldots, z_N) \mod p$ with key K, let f be a public $\mathbb{Z}$-module morphism from $(\mathbb{Z}_p)^N \to (\mathbb{Z}_p)^N$ of matrix $F \in \mathcal{M}_N(\mathbb{Z})$ and let $\mathsf{KS}_{i,j} \in \mathsf{TRLWE}_S(\frac{K_i}{2^j})$ be a key switching key. By applying the functional keyswitch of* Theorem *2 using the integer transformation $F.\mathsf{VDM} \mod p$ whose coefficients are between $[-\frac{p}{2}, \frac{p}{2}]$ to the N extracted TLWE ciphertexts $\mathfrak{c}_i$ of c, we obtain a TRLWE ciphertext c' whose message is $\sum_{i=0}^{N-1} \mu'_i X^i$ where $(\mu'_0, \ldots, \mu'_{N-1}) = \frac{1}{p} f(z_0, \ldots, z_{N-1})$. The noise variance satisfies*

$$\mathsf{Var}(\mathsf{Err}(c')) \leq \left(\frac{Np}{2}\right)^2 \mathsf{Var}(\mathsf{Err}(c)) + \alpha^2 \left(\ell N^2 + \frac{N}{4}\right).$$

TFHE → B/FV. In the reverse direction, we would like to transform k independent TLWE ciphertexts (with messages $(\mu_0, \ldots, \mu_{k-1}) \in \mathbb{T}^k$) into a TRLWE ciphertext with slots $\in \mathbb{Z}/p\mathbb{Z}$. Again, we would need to define a lipschitzian $\mathbb{Z}$-module morphism g between $\mathbb{T}^k$ and $(\mathbb{Z}/p\mathbb{Z})^N$. Unfortunately, since for all $x \in \mathbb{T}^k$, there exists $y \in \mathbb{T}$ such that $x = p.y$, we have $g(x) = p.g(y) = 0$ in $(\mathbb{Z}_p)^N$ and this implies that g is zero everywhere, which is of limited interest.

Therefore, we need to restrict the message space only to multiples of $\frac{1}{p}$ (this prevents division by p). Such a plaintext space restriction may imply that input TLWE ciphertexts must be bootstrapped before exporting them as B/FV slots using gate bootstrapping from Theorem 3.

Then, let g be a morphism from $(\mathbb{Z}_p)^k \to (\mathbb{Z}_p)^N$, of matrix $G \in \mathcal{M}_{N,k}(\mathbb{Z})$. To obtain a B/FV ciphertext whose slots are $g(p\mu_0 \mod p, \ldots, p\mu_{k-1} \mod p)$,

the actual transformation to apply is $\mathsf{VDM}^{-1}.G \mod p$. Again, we can choose the representative with coefficients in $[-\frac{p}{2}, \frac{p}{2}]$, which is $\frac{nP}{2}$-lipschitzian.

If we want to decrease the noise, different possibilities for the algorithm of bootstrapping exist in the literature [7, 17]. The first one is the naive bootstrapping, where we evaluate the rounding function, in this case for $p > 2N + 1$ prime ($p = 1 \mod N$), we need $O(\sqrt{p})$ internal products for the evaluation and we preserve the N slots. The second one is the bootstrapping proposed in [7], where $p = r^e$ is a power of a prime number, and we need only $(e-1)(r-1)$ multiplications, but the number of slots is reduced.

Bridging B/FV-big-number with TFHE. In the case of [9] (based on the NTRU-trick [26]), the plaintext space consists in $\mathbb{Z}[X] \mod (X^N+1) \mod (X-p)$ for a small integer p. By evaluating a message in $X = p$, this space is isomorphic to the ring $\mathbb{Z}/(1+p^N)\mathbb{Z}$ and this allows to evaluate arithmetic operations on big-numbers. In the native plaintext space, $\mathcal{M}$ is composed of all integer polynomial multiples of $\Delta_P(X) = P^{-1} = \frac{-1}{p^N+1}\sum_{i=0}^{N-1} p^{N-1-i}X^i$. Interestingly, since the leading coefficient of the polynomial Δ_P is $1/(p^N+1)$, the isomorphism between $\mathcal{M}$ and $\frac{1}{p^N+1}.\mathbb{Z} \mod 1$ corresponds to extracting the coefficient in $X^N - 1$ (i.e. the mapping $\mu = \sum_{i=0}^{N-1} \mu_i X^i \mapsto \mu_{N-1} \mod 1$). On this native plaintext space, the naïve rounding algorithm: $\mu = \lfloor \varphi.P \rceil.\Delta_P$ can solve the BDD problem up to a distance $\approx \frac{1}{2p}$ (which is the packing radius of the lattice $\mathcal{M}$), which allows to operate on ciphertexts with very large noise $\approx \frac{1}{2\sqrt{N}p}$.

B/FV-big-number → msb-TFHE Given a TRLWE ciphertext $c(X) = (a(X), b(X))$ encoding $\mu(X)$ with a key K, to obtain the TLWE encryption of the most significant bit of $\mu(X)$ with key K it is enough to extract $\mathfrak{c}_{p-1} = \mathsf{SampleExtract}_{p-1}(c(X))$.

TFHE → B/FV-big-number In the inverse direction, to transform $k < N$ TLWE independent ciphertexts $\mathfrak{c}_1, \ldots, \mathfrak{c}_k$ encoding $\mu_i = \frac{x_i}{p}$, where $x_i \in [0, p-1]$ with key S into a TRLWE ciphertext enconding the big-number $R = \sum_{i=1}^{k} x_i p^{N-i} \mod p^N+1$ with key K, we can return an encryption of $(\mu_1, \ldots, \mu_k) \to \sum_{i=1}^{k} \mu_i X^{N-i}$. Indeed, this polynomial is very close to our target $R.\Delta_P \mod 1$. To that end, we can just apply the public key switch $\mathbf{c} = \mathsf{PubKS}(id, \mathsf{KS}, (\mathfrak{c}_1, \ldots, \mathfrak{c}_k))$, where the key switching key is composed by $\mathsf{KS}_i = \mathsf{TRGSW}_K(S_i)$ to pack the k ciphertexts as a single TRLWE ciphertext.

3.3 A general abstraction of HEAAN over the torus

Recently, Cheon et al. proposed HEAAN [11, 12], a homomorphic encryption scheme of approximate numbers based on the RLWE problem. The idea of the scheme is to match the encryption error with the approximation error of the

fixed-point number. In this scheme only the significant digits are used and the phase is taken as a good approximation.

HEAAN is a mixed encryption scheme dedicated to fixed point arithmetic with public exponent. Like scale invariant schemes, the noise is appended to the least significant bits of the phase, but unlike B/FV, the space of valid messages is a small bounded interval, rather than evenly-distributed in the whole space. Also, the maximal amount of noise is interpreted in terms of the ratio between the diameter of the message space and the scale modulus q, rather than the usual noise amplitude versus packing radius. As we keep performing homomorphic operations, the message space diameter increases, until the majority of the space is covered: at this point, the scale invariance property enables to extract the message as a classical LWE sample that can be processed, for instance, by TFHE. To fully enable this bridge between schemes, it is necessary to unify the message spaces. To do so, we revisit the representation of a HEAAN ciphertext with the following three tags/parameters:

- $\rho \in \mathbb{N}$: bits of precision of the plaintext (global constant),
- $\tau \in \mathbb{Z}$: slot exponent (order of magnitude of the complex values in each slot),
- $L \in \mathbb{N}$: level exponent (quantifies the max. amount of homomorphic ops.).

HEAAN can be viewed as an instantiation of TRLWE, whose native plaintext space is the subset of all polynomials $\mu \in \mathbb{T}_N[X]$ of small norm $\|\mu\|_\infty \leq 2^{-L}$. The integer $L > 0$ is the level exponent of the ciphertext, it is public and decreases with each multiplication. When the level is too low, the ciphertext must be bootstrapped to allow further processing. The plaintext space is always given with a global and fixed number ρ of significant digits, so the noise amplitude is implicitly $2^{-(L+\rho)}$. Finally, since the goal is to represent ρ-bit fixed-point values of any order of magnitude, each ciphertext carries a public integer exponent $\tau \in \mathbb{Z}$ which represents the order of magnitude of its slots. Namely, for a given message $\mu \in \mathbb{R}_N[X]$ where $\|\mu\|_\infty \leq 2^{-L}$, its complex slots $[z_1, \ldots, z_{N/2}]$ are the (rescaled) evaluation on the complex roots of $X^N + 1$, so $z_k = 2^{L+\tau}\mu(\zeta_k) \in \mathbb{C}$. The evaluation on the last $\frac{N}{2}$ roots are the conjugates of the first ones, which ensures that μ is real. If $\|\mu\|_\infty \approx 2^{-L}$, this indeed implies that slot values $|z_k| \approx \sqrt{N}2^\tau$, and that the slot precision is up to $2^{\tau-\rho}$.

In order to unify the message spaces, we redefine (tagged) HEAAN ciphertexts as a quadruple $(a(X), b(X), \tau, L) \in \mathbb{T}_N[X]^2 \times \mathbb{Z} \times \mathbb{N}$ where $(a(X), b(X))$ is a TRLWE ciphertext. The error standard deviation is implicitly $2^{-(L+\rho)}$, which corresponds to the parameter α in TFHE. This means that all ciphertexts can safely be rounded to $\approx L + \rho$ bits. As usual, the phase of a ciphertext is $(b(X) - s(X)a(X) \mod 1)$, and the message is the expectation of the phase.

The slots of a ciphertext are the slots of its message $\mu(X)$, interpreted as a small real polynomial in $\mathbb{R}_N[X]$ of norm $\leq 2^L$, so not modulo 1. The slots are the $N/2$ complex numbers $(2^{\tau+L}\mu(\zeta_k))_{k\in(1,\ldots,N/2)}$ (and implicitly, their $N/2$ conjugates). From a matrix point of view, the transformation between the coefficients and the slots is the multiplication with $2^{\tau+L}$ times the complex DFT matrix of ζ_k. We have $(z_1, \ldots, z_{N/2}) = \mathsf{DFT}.(\mu_0, \ldots, \mu_{N-1})$ and $(\mu_0, \ldots, \mu_{N-1}) = 2Re(\mathsf{IDFT}.(z_1, \ldots, z_{N/2}))$.

$$\mathsf{DFT} = \begin{bmatrix} 1 & \zeta_1^1 & \cdots & \zeta_1^{N-1} \\ 1 & \zeta_2^1 & \cdots & \zeta_2^{N-1} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \zeta_{N/2}^1 & \cdots & \zeta_{N/2}^{N-1} \end{bmatrix}, \mathsf{IDFT} = \frac{1}{N} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \bar{\zeta}_1^1 & \bar{\zeta}_2^1 & \cdots & \bar{\zeta}_{N/2}^1 \\ \vdots & \vdots & \cdots & \vdots \\ \bar{\zeta}_1^{N-1} & \bar{\zeta}_2^{N-1} & \cdots & \bar{\zeta}_{N/2}^{N-1} \end{bmatrix} \tag{5}$$

We now express the homomorphic operations over the slots:

Addition $\texttt{HeaanAdd}((\mathbf{c}_1, \tau_1, \mathtt{L}_1), (\mathbf{c}_2, \tau_2, \mathtt{L}_2)) \rightarrow$
$\begin{cases} \mathbf{c} = 2^{\tau_1+L_1-\tau-L}\mathbf{c}_1 + 2^{\tau_2+L_2-\tau-L}\mathbf{c}_2 \mod 1, \\ \tau = \max(\tau_1, \tau_2) + 1, \\ L = \min(L_1 + \tau_1, L_2 + \tau_2) - \tau \end{cases}$

Proof. We can check that this transformation changes the slot value into $2^{\tau+L}\mu(\zeta_k) = 2^{\tau_1+L_1}\mu_1(\zeta_k) + 2^{\tau_2+L_2}\mu_2(\zeta_k) = (z_1 + z_2)$, that proves the correctness of the sum of two slots. The fact that $\tau_i + L_i - \tau - L \geq 0$, for $i \in (1, 2)$ means that the sum is an integer combination of the ciphertexts. At the end we verify that $||\mu(X)||_\infty \leq 2^{\tau_1+L_1-\tau-L}||\mu_1(X)||_\infty + 2^{\tau_2+L_2-\tau-L}||\mu_2(X)||_\infty \leq 2^{\tau_1+L_1-\tau-L-L_1} + 2^{\tau_2+L_2-\tau-L-L_2} \leq 2^{-L}$.

Decrease level $\texttt{HeaanRS}_{\mathtt{L}\rightarrow\mathtt{L}'}((\mathbf{c}, \tau, \mathtt{L}), \mathtt{L}' < \mathtt{L}) \rightarrow (2^{\mathtt{L}-\mathtt{L}'}\mathbf{c}' \mod 1, \tau, \mathtt{L}')$

Proof. We verify that the slot values are preserved, and that $||\mu(X)||_\infty < 2^{L-L'}||\mu'(X)||_\infty \leq 2^{L-L'-L} = 2^{-L'}$.

Binary Shift (multiply by 2^t) $\texttt{HeaanBS}(\mathtt{t}, (\mathbf{c}, \tau, \mathtt{L})) \rightarrow (\mathbf{c}, \tau + \mathtt{t}, \mathtt{L})$

Proof. Slots are indeed transformed as $z' = 2^{\tau+t+L}\mu(\zeta_k) = 2^t z$. Since the ciphertext does not change, the native plaintext does not change either, and the bound 2^{-L} is preserved.

Multiplication with constant $\texttt{HeaanMultCst}(\mathtt{a} \in \mathbb{Z} \quad \text{s.t.} |\mathtt{a}| \leq 2^\rho, (\mathbf{c}, \tau, \mathtt{L}) \rightarrow (\mathtt{a}.\mathbf{c}' \mod 1, \tau + \rho, \mathtt{L} - \rho)$

Proof. We can check that multiplication with $a \leq 2^\rho$ transforms the slot value into $z' = 2^{\tau+L}a\mu'(\zeta_k) = az$, and the bound on the native plaintext becomes $2^{-L+\rho}$. Note that the combination of constant integer multiplication and binary shift allows to multiply by an arbitrary fixed-point plaintext of precision ρ.

Constant slot-wise multiplication $\texttt{HeaanSlotMult}((\mathtt{u}_1, \ldots, \mathtt{u}_{\mathtt{N}/2}), (\mathbf{c}, \tau, \mathtt{L}))$
Let $u_1, \ldots, u_{N/2}$ be N fixed-point complex slots of the same order of magnitude (e.g. $u_k = (x_k + iy_k).2^{-\rho}$ where x_k, y_k are integers in $[-2^\rho, 2^\rho]$. Interpolate (or find by least mean square) an integer polynomial $d(x)$ with coefficients in $[-2^\rho, 2^\rho]$ and t an integer exponent such that the slots of $d(X)2^t$ are all good approximations of $z_1, \ldots, z_n$, up to precision $2^{t-\rho}$. Namely,

$$|d(\zeta_k)2^t - u_k| \leq 2^{t-\rho} \text{ for all } k \in [1, \frac{N}{2}]. \tag{6}$$

Then all we need is to multiply the input ciphertext by $d(x)$ and shift the result by τ bits. The level decreases by ρ bits, where 2^ρ is the norm of d.

$$d(X) \in \mathbb{Z}_N[X](||d||_\infty \leq 2^\rho), (\mathbf{c}, \tau, L) : \begin{cases} d(X) \cdot \mathbf{c} \mod 1, \\ \tau' = \tau + \rho + t, \\ L' = L - \rho. \end{cases}$$

Proof. It follows from (6) that $z'_k = 2^{\tau'+L'}\mu(\zeta_k)d(\zeta_k) = 2^{\tau+L}\mu(\zeta_k)d(\zeta_k)2^t = z_k \cdot (u_k + \varepsilon_k)$ where $|z_k\varepsilon_k| \leq 2^{\tau'-\rho}$ and that the native plaintext norm verifies $||\mu'(X)||_\infty \leq 2^{-L+\rho} = 2^{-L'}$.

Slot-wise precomputed secret multiplication :
`HeaanPrivSlotMult`(TRGSW(D), $(\mathbf{c}, \tau, \mathtt{L})$)
In the previous multiplication, $d(x)$ can be provided encrypted as a TRGSW ciphertext of D.

General multiplication `HeaanMult`$((\mathbf{c}_1, \tau_1, \mathtt{L}'), (\mathbf{c}_2, \tau_2, \mathtt{L}'))$ Use the Algorithm 1 below, proved in Proposition 2.

Algorithm 1 HEAAN homomorphic product on $\mathbb{T}_N[X]$

Input: Two HEAAN ciphertexts (a_1, b_1, τ_1, L_1), $(a_2, b_2, \tau_2, L_2) \in \mathbb{T}^2 \times \mathbb{Z} \times \mathbb{N}$ whose slots are $(z_{1,1}, \ldots, z_{1,N/2})$ and $(z_{2,1}, \ldots, z_{2,N/2})$ under the same key s and precision $\rho > \log_2(N)$.

Output: a HEAAN ciphertext (a, b, τ, L) whose slots are $z_j = z_{1,j}z_{2,j}$ for $j \in [1, N/2]$ with the same key s

1: Set $\tau = \tau_1 + \tau_2$ (slot exponent)
2: Set $L' = \min(L_1, L_2)$ and use $\mathtt{HeaanRS}_{\mathtt{L_i} \to \mathtt{L}'}$ to decrease both ciphertexts to level L'
3: Let $q = 2^{L'+\rho}$, $\alpha = \frac{1}{q}$, and $L = L' - \rho$
4: Round (a_i, b_i) to the nearest multiple of $\alpha = \frac{1}{q}$.
5: Let $(a, b) = (a_1, b_1) \boxtimes_{q,\alpha} (a_2, b_2)$ (with $\boxtimes_{q,\alpha}$ the internal homomorphic product defined in the Definition 2)
6: **return** (a, b, τ, L)

Proposition 2 (HEAAN product). *Let (a_1, b_1, τ_1, L_1), $(a_2, b_2, \tau_2, L_2) \in \mathbb{T}_N[X]^2 \times \mathbb{Z} \times \mathbb{N}$ whose slots are $(z_{1,1}, \ldots, z_{1,N/2})$ and $(z_{2,1}, \ldots, z_{2,N/2})$ under the same key s. We suppose that the precision ρ is larger than $\log_2(N)$. Algorithm 1 computes a HEAAN ciphertext (a, b, τ, L) whose slots are $z_j = z_{1,j}z_{2,j}$ for $j \in [1, N/2]$ with the same key s such that $\mathsf{Var}(\mathsf{Err}((a, b)))$ remains implicitly $4^{-L-\rho}$.*

Proof. (sketch) Since Algorithm 1 rescales both ciphertexts to the same level, we can assume that both inputs have the same level L'. Compared to the proof of Lemma 2, defining the same auxiliary quantities C_0,C_1,C_2, we have

$$\begin{aligned} \varphi_s(a, b) &= \mu_1 \boxtimes_q \mu_2 + e_2\mu_1 q + e_1\mu_2 q + e_1 e_2 q + e_2 I_1 q + e_1 I_2 q \\ &\quad + \mathsf{Err}(RK \boxdot_\alpha (C_2, 0)) \mod 1 \\ &= \mu_1 \boxtimes_q \mu_2 + e_2\mu_1 q + e_1\mu_2 q + e_1 e_2 q + \mathsf{Err}(RK \boxdot_\alpha (C_2, 0)) \mod 1 \end{aligned}$$

Here, the terms $e_1 I_2 q$ and $e_2 I_1 q$ dissapear because e_i are exact multiples of $\frac{1}{q}$ after the rounding. The expectation of the phase is still $\mu_1 \boxtimes_q \mu_2$, so the output slots contain $z_k = q\mu_1(\zeta_k)\mu_2(\zeta_k)2^{\tau+L} = z_{1,k}z_{2,k}$ since $L = 2L' - \log_2(q)$. The native plaintext $\mu_1 \boxtimes_q \mu_2$ itself is bounded by $q2^{-2L'} = 2^{L'+\rho-2L'} = 2^{-L}$. The phase variances of $e_2\mu_1 q$, $e_1\mu_2 q$ are bounded by $\left(q2^{-L'}2^{-L'-\rho}\right)^2 = 4^{-L-\rho}$, $e_1 e_2 q$ by $4^{-L-2\rho}$, and $\mathsf{Var}(\mathsf{Err}(RK \boxdot_\alpha (C_2, 0)) \leq \left(2\ell N + \frac{N^2+N}{4}\right)\alpha^2 \leq N^2\alpha^2 \leq 4^{\log_2(N)-L'-\rho} < 4^{-L'} \leq 4^{-L-\rho}$ because $\rho > \log_2(N)$. Overall, the output noise standard deviation is $2^{-L-\rho}$, which corresponds to ρ bits of fixed-point precision. □

TFHE → HEAAN Given some TLWE ciphertext that encrypts μ_i, the first use-case that we may imagine is to produce a HEAAN ciphertext whose slots contain μ_i. However, μ_i is defined modulo 1, and the slots are over $\mathbb{C}$, so to define it properly, we would need to define a canonical representative, like for instance the one in $[-\frac{1}{2}, \frac{1}{2}]$. Overall, this requires to homomorphically evaluate the mod 1 operation, which is essentially the bootstrapping proposed by HEAAN in [11] combined with multiplications with DFT matrix in order to switch between coefficients and slots.

Another point of view is that given $N/2$ TLWE ciphertexts of $\mu_1, \ldots, \mu_{N/2}$, we may instead want to get a single HEAAN ciphertext having $\exp(2i\pi\mu_k)$ with level L inside its slots. This allows to evaluate trigonometric polynomials, which have a lot of potential applications, for instance for evaluating continuous functions via fast-convergent Fourier series. We describe this algorithm as a variant of the bootstrapping for HEAAN [11].

Proposition 3 (Functional switching TFHE to HEAAN). *Given $N/2$ TLWE ciphertexts $(a_1, b_1), \ldots, (a_{N/2}, b_{N/2})$ of $\mu_i \in \mathbb{T}$ with the key S, $BK_i = \mathsf{TRGSW}_K(S_i)$ with noise standard deviation α, and a precision parameter $\rho \in \mathbb{N}$, Algorithm 2 computes a HEAAN ciphertext (a, b, τ, L) whose slots are $z_k = \exp(2i\pi\nu_k)$, where $\nu_k = \varphi_S(a_k, b_k)$ with a key K, with precision ρ, $p = \sqrt{\rho} + \log_2(\frac{2\pi n}{\sqrt{\rho}})$ and*

$$L = -\log_2 \alpha - (p + \log_2 \rho)\rho - \frac{1}{2}\left(\log_2\left(-2nN\log_2\alpha + n\frac{1+N}{4}\right)\right)$$

Proof. (sketch) To approximate $\exp(2\pi i\nu_k)$ we used the idea of [11]: we first take a small real representative of the input ciphertexts, and divide them by 2^p for a suitable p (that depends on the target precision). This way, the (real) phase of the rescaled ciphertext $\nu_k/2^p$ is guaranteed to be bounded $n/2^p$. We first compute a good approximation, up to an error $\leq 2^{-\rho}$ for $\exp(2\pi i\nu_k/2^p)$ using the first $\sqrt{\rho}$ terms of the Taylor expansion of exp. For instance, Taylor-Lagrange inequality gives $|\exp(ix) - (\sum_{k=0}^{\sqrt{\rho}-1} \frac{(ix)^k}{k!})| \leq \frac{|x|^{\sqrt{\rho}}}{\sqrt{\rho}!}$, so for $x \leq n/2^p$, it suffices to

Algorithm 2 Switching TFHE to HEAAN

Input: $N/2$ TLWE ciphertexts (a_k, b_k) whose phases are $\nu_k \in \mathbb{T}$ under the same key $\mathbf{s} \in \mathbb{B}^n$ and $BK_i = \mathsf{TRGSW}_{\mathsf{K}}(s_i)$.

Output: a HEAAN ciphertext $(a(X), b(X))$ at level L whose slots are $z_k = e^{2\pi i \nu_k}$ for $k \in [1, N/2]$ with key K.

1: We call A the $N/2 \times n + 1$ real matrix where $A_{i,j}$ is the representative of the j-th coefficient of $a_i \in [-\frac{1}{2}, \frac{1}{2}]$, and in the last column $A_{j,n+1}$ contains the representative of $b_j \in [-\frac{1}{2}, \frac{1}{2}]$.
2: Let $p = \sqrt{\rho} + \log_2(2\pi n/\sqrt{\rho})$
3: Compute $P_j \leftarrow \frac{1}{2^p N}\mathrm{Re}(\mathsf{IDFT} * A_j)$ for $j \in [1, n+1]$. (P_j is the polynomial whose slots are $\frac{1}{2^p}A_j$).
4: $\mathbf{c} \leftarrow (0, 2^{-(L+(p+1)\rho)}P_{n+1}) - \sum_{j=1}^{n} BK_j \boxdot_\alpha (0, 2^{-(L+(p+1)\rho)}P_j)$.
5: Let $C = (\mathbf{c}, \tau = 0, L + (p+1)\rho)$
6: Evaluate homomorphically $E = \sum_{k=0}^{\sqrt{\rho}-1} \frac{i^k}{k!} C^k$ using Paterson algorithm (in depth: $\log_2(\rho)$), `HeaanMult` for non-constant multiplications, and `HeaanSlotMult` for constant multiplications. (E has parameters $\tau = 0$ and level $L + p\rho$)
7: **for** $j = 1$ to p **do**
8: $\quad E \leftarrow \mathtt{HeaanMult}(\mathtt{E}, \mathtt{E})$ (the new E has parameters $\tau = 0$ and level $L + (p-j)\rho$)
9: **end for**
10: **return** E at level L

choose $p = \sqrt{\rho} + \log_2(\frac{2\pi n}{\sqrt{\rho}})$ to get the required approximation within $2^{-\rho}$. Then, we square and multiply (we square and square in this case) the result to raise $\exp(2\pi i \nu_k / 2^p)$ to the power 2^p to obtain the desired plaintext $\exp(2\pi i \nu_k)$.

From Theorem 1 on TFHE's external product, the noise of the ciphertext $\mathbf{c}$ of line 4 is $\mathsf{Var}(\mathsf{Err}(\mathbf{c})) \leq (-2n \log_2 \alpha N + n\frac{1+N}{4})\alpha^2 \leq 4^{-L-(p+\log_2(\rho))\rho}$.

Then, we evaluate the Taylor expansion of the complex exponential (up to degree $d = \sqrt{\rho}$) via Patterson algorithm: this requires a depth of $2\log_2(d) = \log_2(\rho)$, uses $3\sqrt{d}$ non-constant (`HeaanMult`), and d constant multiplications (`HeaanSlotMult`). After this step, the level decrease by ρ times the multiplicative depth, so the level of E is $\leq L + p\rho$.

Finally, we square the ciphertext p times to obtain the desired result. □

HEAAN → TFHE In the reverse direction, once we have decreased the level L of a HEAAN ciphertext to 1, so that the native plaintext covers the whole torus interval, we can directly use the slots to coefficients procedure described in [11] to extract HEAAN slots into coefficients of TRLWE (i.e. applying the IDFT complex transformation homomorphically).

4 Tools for homomorphic neural network evaluation

In the previous section, we showed how to switch between different schemes. We focus now on non-linear operations, such as the absolute value or the sign function. Non-linear functions play a crucial role in homomorphic neural network

evaluation as they permit among others to perform homomorphic comparisons, to compute a maximum, to evaluate piecewise functions (e.g. the ReLU activation function), or to evaluate a rounding, or a decryption function.

Note that the ReLU and max are easily expressed with the absolute value: for x, y in $(-\frac{1}{4}, \frac{1}{4})$, $2\max(x, y) = (x + y) + |x - y|$, and for 2.$\mathtt{ReLU}(x) = x + |x|$.

4.1 Non-linear functions in TFHE

In TFHE, non-linear functions from $\mathbb{T} \to \mathbb{T}$ can be homomorphically applied to the phase of an individual TLWE ciphertext via the functional bootstrapping (Theorem 3). The constraints in this case are the following. The domain of the function is restricted to only multiples of $1/2N$ where N is the bootstrapping key size (in particular, it is a medium-sized power of 2), and the function must be negacyclic, i.e. $f(x + \frac{1}{2}) = -f(x)$. Otherwise, the function is defined pointwise, so its graph can be arbitrary.

In the case of the absolute value and the sign function, there exists, as shown below, a change of variables that makes both these functions negacyclic. For other functions, if such a variable change does not exist, the domain of the function should be restricted to half of a torus.

For instance, in the case of the absolute value function, suppose c is a TLWE ciphertext that encrypts $\mu \in \mathbb{T}$ and let c' be the encryption of the absolute value $|\mu|$. To compute c' it is enough to use the gate bootstrapping algorithm to evaluate the negacyclic function $f(x) = |x| - \frac{1}{4}$ and to translate the result by adding a trivial ciphertext of $\frac{1}{4}$ (see Figure 3). After bootstrapping, the message of c' is exactly $|\mu|$ but of course, the phase $\varphi_s(c')$ is within a small Gaussian error around $|\mu|$. In Section 5, we use this noise model to estimate the stability of neural networks if the ReLU is evaluated via TFHE's gate bootstrapping.

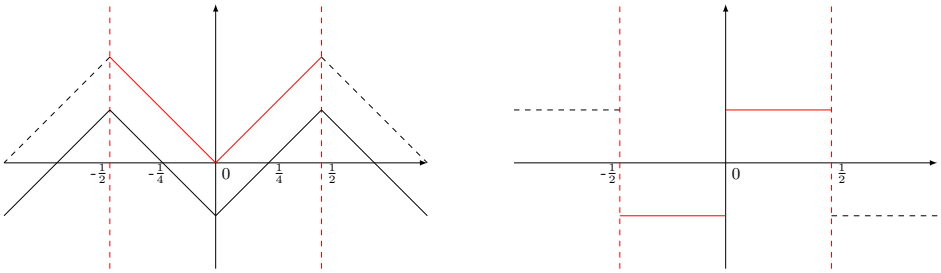

Fig. 3: Absolute (on the left) and Sign (on the right) values TFHE

4.2 Non-linear functions in B/FV

In B/FV any arbitrary function from $\mathbb{Z}_p$, for a prime p, to itself is a polynomial of degree $\leq p-1$, and can be evaluated as an arithmetic expression of multiplicative

depth $2\log_2(p)$. This evaluates the polynomial simultaneously on the N slots. If the function is described from $\mathbb{T} \to \mathbb{T}$, both the domain and the image are in this case rounded to exact multiples of $\frac{1}{p}$. Compared to TFHE, the output is more constrained (e.g. impossible to output non-multiples of $\frac{1}{p}$), but on the other hand, there is no negacyclic constraint on the input, so the graph of the non-linear function can be arbitrary everywhere.

However, except in very special circumstances, the polynomial to evaluate is dense, so the number of homomorphic operations is $\Theta(p)$, which prevents from using large p: the method does not apply to big-number slots. One notable exception to this rule is the bootstrapping in [7] modulo p^k, which proves that the rounding function is the composition of sparse polynomials.

4.3 Non-linear functions in HEAAN

In HEAAN, non-linear functions can be obtained either as complex polynomials (classical approach), or as trigonometric polynomials (via Algorithm 2 that packs complex exponential values, and was also used in HEAAN's bootstrapping). Fourier series of smooth and regular functions converge very quickly: for example, the Fourier series of a C^∞ function converges super-algebraically fast, and if one smooths any periodic function by convolution with a small Gaussian, its Fourier series converges super-exponentially fast. However, the convergence is slower if the function has discontinuities (convergence in $\Omega(1/d)$), or discontinuities in its derivative (convergence in $\Omega(1/d^2)$) where d is the degree of the serie.

For example, the absolute value coincides with the triangular signal $(-\frac{1}{2}; \frac{1}{2})$, which extends naturally a 1-periodic continuous function (piecewise C^1). Given $N/2$ TLWE ciphertexts, we can efficiently pack the complex exponential of their phases $\exp(2i\pi\mu)$ in the slots of a single HEAAN ciphertext. Subsequently, we can evaluate any trigonometric polynomial of small degree, and extract the results back to TLWE samples. For instance, the triangle (corresponding to the absolute value) has the following Fourier sequence with only cosinus terms of odd degrees, and that converge in $O(d^2)$, and the square signal (corresponding to the sign or decryption function has only sinus terms of odd degrees).

$$\mathrm{Abs}(x) = K_1 \sum_{k=0}^{\infty} \frac{\cos 2\pi(2k+1)x}{(2k+1)^2} + K_2 \tag{7}$$

$$\mathrm{Sign}(x) = K_1 \sum_{k=0}^{\infty} \frac{\sin 2\pi(2k+1)x}{(2k+1)} + K_2 \tag{8}$$

Figure 4 shows that the first three (resp. six) terms of the Fourier series of the absolute value and the Sign function already form a good approximation on the interval $\left[-\frac{1}{2}, \frac{1}{2}\right]$.

Compared to classical approximations of functions by polynomials in [21, 6] (i.e. Taylor series or Weierstrass theorem), Fourier series have three main advantages: they do not diverge to ∞ outside of the interval (better numerical stability), the Fourier coefficients are small (square integrable), and the series

converge uniformly to the function on any interval that does not contain any discontinuity in the derivative. However, in the particular case of $Abs(x)$ and $Sign(x)$, the presence of a singularity or discontinuity in 0 in both graphs implies that the series converge poorly around 0. Unfortunately, native plaintexts in HEAAN ciphertext at level L have by definition tiny phases in the interval $\left[-\frac{1}{2^L}, \frac{1}{2^L}\right]$. We address this problem using the bootstrapping capability of HEAAN: First, use $\texttt{HeaanRS}_{\texttt{L}\to\texttt{0}}$ to decrease the level $L = 0$ or $L = 1$, so that input phases range over a large torus interval $\left(-\frac{1}{2}, \frac{1}{2}\right)$ or $\left(-\frac{1}{4}, \frac{1}{4}\right)$, and then, divide K_1 by 2^L so that the output has level L.

With this bootstrapping trick, HEAAN can at the same time evaluate a non-linear function and bootstrap its output to a level L even higher than its input. Taking this fact into account, instead of writing $\texttt{ReLU}(x) = \max(0, x)$ as $\frac{1}{2}(|x| + x)$ like in TFHE or B/FV, where the term $+\frac{x}{2}$ is not bootstrapped, it is actually better to extend the graph of `ReLU` from a half period $(-\frac{1}{4}, \frac{1}{4})$ directly to a 1-periodic continuous function, and to decompose the whole graph as a Fourier series. In the latter case, the output level L can be freely set to an arbitrary large value. Figure 4 shows a degree-7 approximation of the odd-even periodic extension of the graph of $\texttt{ReLU}(x)$.

If the `ReLU` is evaluated via this technique, the output message is the Fourier approximation, and the phase still carries an additional Gaussian noise on top of it. In the next section, we also study the robustness of neural networks with this approximation and perturbation model.

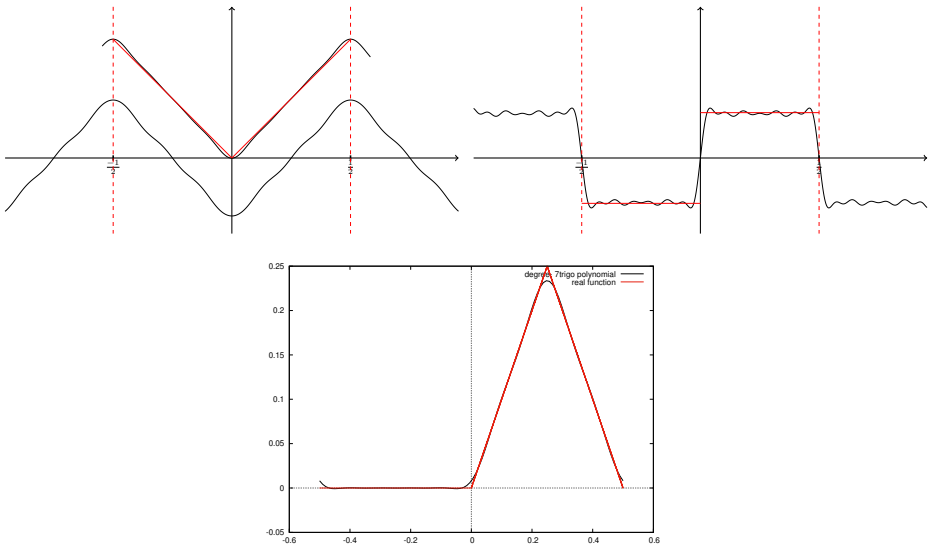

Fig. 4: Absolute value (on the left), Sign value (on the right) and `ReLU` (sbottom) for HEAAN

5 Application to Deep Learning

Neural networks (NN) are computing systems composed of simple interconnected processing units, called *neurons*, trained to solve among others classification problems. Connections between neurons have weights that adjust as learning proceeds. Neural networks are organized in layers. Networks with multiple layers are known as *deep*. *Convolutional neural networks* (CNNs) are a special type of deep neural networks that have been proven very successful in image recognition and classification. CNN inputs are images organized in three dimensional arrays of pixels (one two-dimensional array per colour channel) and are typically composed of five types of operations.

Preserving the privacy of sensitive (e.g. medical, financial, ...) data while applying machine learning algorithms and still ensure good performance and high output accuracy is lately a high-interest problem for both the cryptographic and the maching learning community [6, 22, 2]. We briefly describe now the main layers composing a CNN from a FHE point of view.

Convolution The purpose of this operation is to extract features from the input image (such as lines or borders). This is achieved by computing element-wise products of two matrices, the first one being a submatrix of the input matrix and the second-one being some filter. The weights within the filter are learned in plaintext during the training and are subsequently encrypted. During the evaluation phase, convolution is a secret affine function, which can be efficiently evaluated using the TRGSW-TRLWE external product in the three scenarios (TFHE, B/FV and HEAAN) of our unified framework.

Nonlinearity An activation function is then applied to the output of the convolution step. The purpose of this layer is to introduce nonlinearity and nowadays it is always almost achieved by the `ReLU` (REctified Linear Units) function $f(x) = \max(0, x)$ [23]. The homomorphic evaluation of the `ReLU`, as well as its noise model have been studied for the three scenarios in the previous section. In almost all previous works, the standard approach was to replace the `ReLU` by a function with a lower multiplicative depth. In [22], `ReLU` is notably approximated by the square function $f(x) = x^2$, in [2] it is replaced by the sign function, while in [6] the `ReLU` is approximated by low-degree polynomials.

Pooling This layer reduces the dimensions of the input by retaining the most important information. The image is partitioned into non-overlapping sub-matrices and for each sub-matrix a single information is retained. This is typically done by computing the maximum value and this procedure is called *max pooling*. Other types of pooling exist, such as for example the *average pooling* that computes the average of the elements in the concerned region. This layer by reducing the size of the matrices, permits to reduce the size of the parameters and to regulate overfitting. Computing the maximum of two values can be achieved via the absolute value, as shown in the last section. Yet, no efficient algorithm is known to compute the maximum of a large number of values. On the contrary, average pooling is linear with

public coefficients and therefore FHE-friendly. In [22] the authors replace max pooling by sum pooling, while in [6] max pooling is replaced by average pooling.

Fully Connected (FC) Layer All the neurons of this layer all connected to all neurons of the previous layer. Their activation is computed by a matrix multiplication plus a bias offset. This is again a secret precomputed affine step that can be achieved via the external product.

Loss Layer This is normally the last layer of a CNN and it's role is to constantly compare the guesses of the fully-connected layer with the actual values with the aim to minimize the differences. During training, the loss layer is represented by a continuous cost function (using e.g. a sigmoid) that quantifies by how much the current model mis-classifies the training set and the weights in the whole network are adjusted by gradient descent on this cost function. During the evaluation, the loss layer becomes an argmax operation. This last step is in general ignored in other homomorphic implementations of neural networks. For example in [22, 2] the authors simply output the score vectors, and the recipient computes the best one after decryption. To do this final step homomorphically, the Boolean approach of TFHE seems to be the most suited to this non-SIMD step.

5.1 Robustness against the FHE error models

In this section, we simulate the homomorphic execution of the neural network by replacing the value output of each non-linear layer by a random sample which has the same distribution as the phase of TRLWE samples after a homomorphic evaluation of the layer. This approach allows us to simulate a homomorphic evaluation, and to obtain accurate predictions on the outcome without having to the run the expensive homomorphic computation. This allows to estimate the largest noise standard deviation α that can be tolerated by the network, and therefore, the smallest FHE parameters required to evaluate it.

As explained above, in the context of FHE, the training of networks is usually done on the plaintexts without any perturbations occurring, and only then, the network is encrypted to the cloud to protect the privacy of the model during predictions. In this direction, we carried out many experiments on three different convolutional neural networks structures, using the TFHE and HEAAN noise models of Section 4, in order to measure their robustness against such perturbations.

This approach is not new. For example, in [10] the authors studied the stability of CNNs by applying among others a Gaussian perturbation to the internal weights inside the convolutional layers. The applied Gaussian was centered at zero and had a standard deviation relative to the standard deviation of that layer's original weight distribution. This type of perturbation modifies the average value of the inputs to the convolutional layer. Even, if the motivation of this paper is not linked to homomorphic computations, their conclusions and ours intersect at some points. Indeed, the authors of [10] noticed that the last convolutional layers are surprisingly stable, while the first convolutional layers

are much more fragile and so the accuracy depends on the level the perturbation applies. The most surprising result that we obtain in our experiments is that all the neural networks we tested support quite large relative errors of at least 10%, without any impact on the global accuracy. In a TFHE context, raising the error amplitude from a usually required 2^{-40} negligible amount to 2^{-4} means that the depth of leveled TRGSW circuits (number of transitions in automata in leveled circuits in [13]) can be increased by a factor $(2^{36})^2$ (i.e. unlimited) without changing the parameter sets. This also means that only 4 bits of precision (instead of 20 to 40 bits usually) are needed on all fixed point operations throughout the network, which results in very small parameter sets for HEAAN or B/FV.

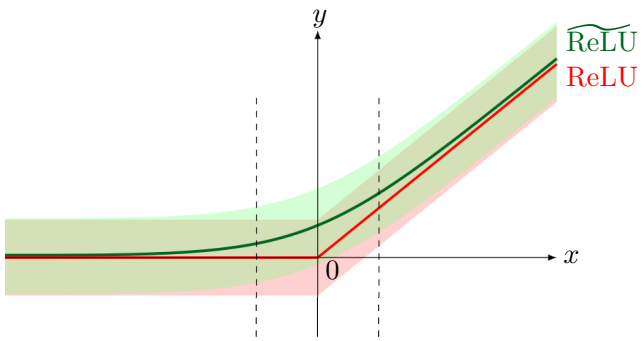

Fig. 5: Noise distributions around the `ReLU` function

5.2 Experiments

We conducted experiments with three different convolutional neural networks and for all of them we used the `dlib C++` library [27]. The first network is LeNet-5 [28], that can be trained to recognize handwritten digits, the second-one is a 9-layer CNN trained to distinguish cat and dog pictures, and the last one is the ResNet-34 network [25], a deep network of 34 layers able to classify an input image into one of 1000 objects. We briefly describe each of the networks and the experiments done on it.

LeNet-5: Recognition of handwritten digits LeNet-5 is a well-known convolutional 7-layer neural network designed by LeCun et al. in 1998 to recognize handwritten digits [28]. In the original version of the network, the sigmoid was used as the activation function. In the version that we manipulated, implemented in the `dlib` library [27], the `ReLU` activation function is used instead.

We trained this network on the MNIST dataset [29], composed of 60000 training and 10000 testing images, with two different versions of the pooling algorithm. We first trained the network by using max pool for both pooling layers and at a second stage we re-trained it from scratch by replacing now max

pool by average pool. Our goal was to see how each version reacts to perturbations. In particular, we added to each output value of the activation function a value drawn from a Gaussian distribution with mean value zero and some standard deviation σ. This was done for the activation function of all levels. For our experiments we further used two different activation functions: the original `ReLU` activation function and then an approximation of the `ReLU` function by a trigonometric function, depicted in green in Figure 5, which can be used in HEAAN as a replacement of $\max(0, x)$. Finally, we perturbed the output of the activation function in two different ways. First by a Gaussian distribution of fixed standard deviation σ and in a second experiment by a standard deviation proportional to the input's standard deviation (which can be publicly estimated during training).

The results of these experiments are summarized in Table 2 of Appendix A and Figure 7. In this example, we pushed standard deviation from 0.0 to 1.0 for both trained CNNs, the one trained with max pool and the other one trained with average pool. In this table we give both the accuracy on the testing set but also on the training set. In order to correctly interpret the right part of Figure 7 it has to be noted that the mean value of the `ReLU` entries was measured between 0.4 and 1.91 and the standard deviation between 0.97 and 2.63.

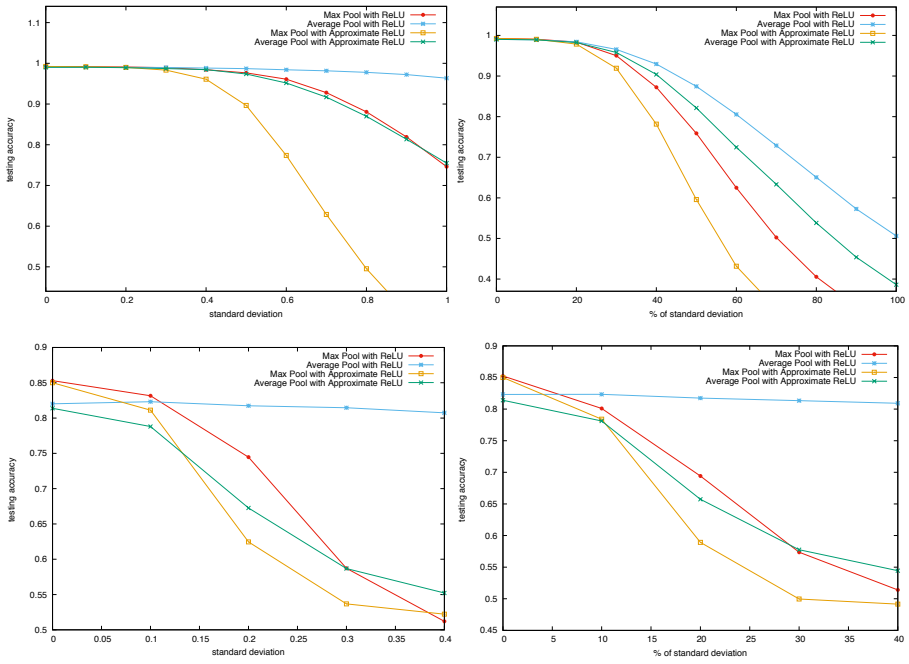

Fig. 7: Experiments with LeNet-5 (up) and the cat versus dog classifier (down). The results with proportional perturbations are on the right, while with non-proportional perturbations on the left.

The first remark that can be done by looking into these experiments is that average pool is much more stable to perturbations than max pool and provides a high accuracy even for large values of the standard deviation. The second remark concerns the accuracy when an approximation of the `ReLU` function is used instead of the original one. As it can be seen from the left part of Figure 7, the accuracy for the average pool version is clearly lower when a `ReLU` approximation is used, but still has a very good score (over 95%) for standard deviations as high as 0.6. Finally, special care has to be taken when interpreting the results corresponding to the application of a proportional perturbation of the input data standard deviation. In the right part of Figure 7 the x-axis corresponds to a perturbation equal to the percentage of the inputs' standard deviation. Depending on the original deviation of the input distribution, the perturbation can be extremely important and this is why the accuracy shows to drop. Therefore, one has to keep in mind that the perturbation of the right-side figures is in general more important and probably also more meaningful than the one of the left-side figures.

Cats versus Dogs Classifier In this section we present our results and remarks on a simple 9-layer neural network that was trained to classify pictures as cats or dogs. For this, we used again the `dlib` library [27] and coded with it the 9-layer NN presented in [1]. The structure of this NN is depicted in Figure 8. This network is composed of 3 convolution layers followed by the `ReLU` activation function, two fully connected (FC) layers and two pooling layers. In the original net, the max pool operation is used at this step. The 7-th layer is a dropout layer, that is a standard technique for reducing overfitting and consists in ignoring a different randomly chosen part of neurons during the different stages of the training phase [31]. We trained this network on the Asirra dataset [16] used by Microsoft Research in the context of a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) challenge. Most of the good CNNs trained to distinguish dogs from cats achieve more than 80% accuracy on the testing set while the accuracy on the training set is usually around 100%. The difference in the two performances is usually due to some overfitting occuring.

We did exactly the same type of experiments for this network and the results can be found in Table 3 of Appendix B or visualized in the lower part of Figure 7. This network is a little-bit more complex than LeNet-5 and seems to be less stable. For this reason, the higher standard deviation considered here is 0.4. However, globally, the same remarks as for LeNet-5 network result. Again, for correctly interpreting the right part of the table, it has to be noted that the mean value of the inputs of the activation function ranges between 0.0004 and 0.628 and the standard deviation ranges between 0.0067 and 3.51.

ResNet-34 ResNet (Residual Network) is a recent family of very deep convolutional neural networks showed to perform extremely well [25]. The global layer structure is very similar to a classical CNN, however better performances are

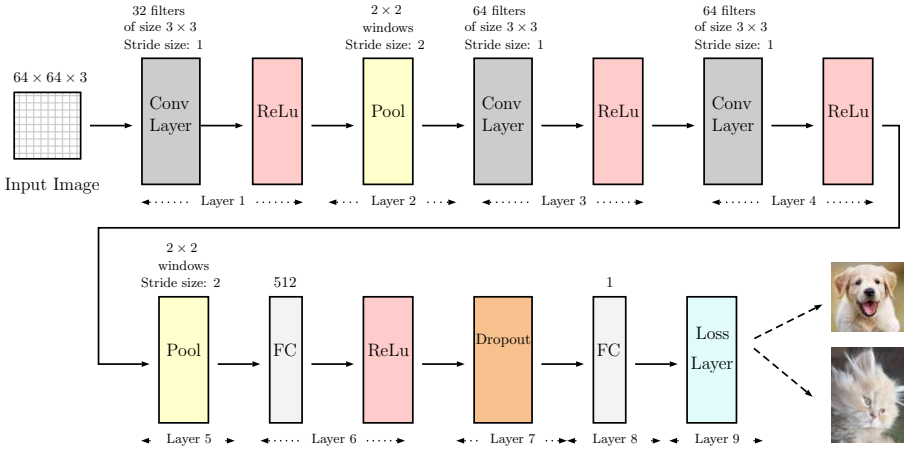

Fig. 8: 9-layer neural network [1] trained to classify pictures as cats or dogs.

achieved by the introduction of a shortcut connection, that consists in skipping one or more layers. The version that we used is composed of 34 layers, and is usually abbreviated as ResNet-34. This network, once trained, is able to classify photos of objects into 1000 distinct object categories.

The training of such residual networks is extremely time consuming (two weeks on a 16-GB Titan GPU, and about 20 times more on 16-CPU cores) and because of time constraints we were not able to finish the training on a network where max pooling is replaced by average pooling. Thus, we were only able to perform our experiments on the pre-trained network on the imagenet ILSVRC2015 dataset [30] and the results are reported in Table 1. Top 1 and Top 5 labels report respectively the percentage of the pictures in the validation set that were correctly classified (Top 1) and whose correct label appeared in the five top suggestions provided by the network (Top 5).

		Non-proportional perturbation				Proportional perturbation			
		`ReLU`		$\widetilde{\texttt{ReLU}}$		`ReLU`		$\widetilde{\texttt{ReLU}}$	
Pool type	σ	Top 1	Top 5	Top 1	Top 5	Top 1.	Top 5	Top 1	Top 5
max	0.0	0.7416	0.9158	0.7428	0.9187	0.7439	0.9202	0.7398	0.9166
max	0.1	0.7357	0.9132	0.7056	0.9165	0.7132	0.8948	0.7586	0.9252
max	0.2	0.6991	0.8860	0.7056	0.8967	0.1562	0.3294	0.3658	0.6027
max	0.3	0.5068	0.7267	0.4829	0.7171	0.0019	0.0089	0.0012	0.0079
max	0.4	0.1500	0.0498	0.1065	0.0817	0.0018	0.0085	0.000	0.0009
max	0.5	0.0233	0.0608	0.0017	0.0066	0.0001	0.0044	0.000	0.0010

Table 1: Experiments on ResNet-34 with max pooling and with perturbations of standard deviation ranging from 0.0 to 0.5. The right columns correspond to perturbations proportional to the input's standard deviation.

5.3 Conclusion/Discussion

Finally, we discuss the lessons learned from the experiments with the three different CNNs, provide links with the developed theory and especially with Section 4 and give concrete guidelines on which type of operation should be performed with which fully homomorphic scheme depending on the given use case.

Max pool versus Average pool We conducted experiments on LeNet-5 and the 9-layer CNN classifying cats and dogs, by replacing during the training and the evaluation the classical max pooling operation by the average pool. This modification, applied also in [6] and to some extend in [22], offers a significant advantage for all FHE schemes, as this operation is affine with public coefficients, compared to max pool that is non-linear. Our experiments showed that this approach offers a further advantage in FHE, as it is way more stable than max pool to perturbations. This behaviour has a natural mathematical explanation, since the standard deviation of an average of independent samples is smaller than the input standard deviations.

Proportional versus non-proportional perturbations We applied two types of perturbations to all three networks. The first type of perturbations was the addition at the output of the activation function of a value drawn from a Gaussian distribution with zero mean and a fixed standard deviation. In the second type of perturbations, the value added had a standard deviation proportional to the standard deviation of the input distribution. The second scenario corresponds to the HEAAN fixed-point arithmetic model, where the public plaintext exponent tag τ is set to match the amplitude during the training phase, and therefore, the noise α is by definition relative to 2^{τ}. Surprisingly, without impacting the result, neural networks are able to absorb very large relative errors between 10% and 20% after each `ReLU` (there are respectively thousands, millions, and billions of them in the three tested networks). This means homomorphic parameters need only to ensure $\rho = 4$ bits of precision (HEAAN), or $\alpha = 0.1$ on the native plaintext, instead of the usually recommended 2^{-40}.

Approximating the `ReLU` activation function The main source of non-linearity of a convolutional neural network is coming from the `ReLU` function. In TFHE or B/FV, these functions are evaluated exactly either as circuits, or as pointwise-defined arbitrary functions. Approximating the `ReLU` by something easier is thus a natural approach [22, 6, 2]. In HEAAN such continuous functions can be approximated accurately by low degree trigonometric polynomials. In our experiments with ResNet-34 (see Table 1) the output accuracy is surprisingly even better with an approximated `ReLU` of this type than with the classical one, in the presence of small noise, which proves that this approach is realistic.

References

1. Cats and dogs and convolutional neural networks. http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-

convolutional-neural-networks, September 2016.
2. F. Bourse, M. Minelli, M. Minihold, and P. Paillier. Fast homomorphic evaluation of deep discretized neural networks. *IACR Cryptology ePrint Archive*, 2017:1114.
3. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, 2012.
4. Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *CRYPTO 2016, Proceedings, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, 2016.
5. M. Brenner, W. Dai, S. Halevi, K. Han, A. Jalali, M. Kim, K. Laine, A. Malozemoff, P. Paillier, Y. Polyakov, K. Rohloff, E. Savaş, and B. Sunar. A standard api for rlwe-based homomorphic encryption. Technical report, HomomorphicEncryption.org, Redmond WA, July 2017.
6. H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.
7. H. Chen and K. Han. Homomorphic lower digits removal and improved FHE bootstrapping. In *EUROCRYPT 2018, Proceedings, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, 2018.
8. H. Chen, K. Laine, and R. Player. Simple encrypted arithmetic library - SEAL v2.1. In *Financial Cryptography and Data Security - FC 2017*, pages 3–18, 2017.
9. H. Chen, K. Laine, R. Player, and Y. Xia. High-precision arithmetic in homomorphic encryption. In *CT-RSA 2018*, volume 10808 of *LNCS*, pages 116–136. Springer, 2018.
10. N. Cheney, M. Schrimpf, and G. Kreiman. On the robustness of convolutional neural networks to internal architecture and weight perturbations. *CoRR*, abs/1703.08245, 2017.
11. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2018, Proceedings, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, 2018.
12. J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017, Proceedings, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, 2017.
13. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption over the Torus. *IACR Cryptology ePrint Archive*, 2018:421.
14. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016, Proceedings, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, 2016.
15. L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015, Proceedings, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, 2015.
16. J. Elson, J. R. Douceur, J. Howell, and J. Saul. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, pages 366–374. ACM, 2007.
17. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
18. M. Geihs and D. Cabarcas. Efficient integer encoding for homomorphic encryption via ring isomorphisms. In *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 48–63. Springer, 2015.

19. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.
20. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013, Proceedings, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, 2013.
21. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210.
22. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.
23. R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947, June 2000.
24. S. Halevi and V. Shoup. Algorithms in HElib. In *CRYPTO 2014, Proceedings, Part I*, volume 8616 of *LNCS*, pages 554–571. Springer, 2014.
25. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR 2016*, pages 770–778. IEEE Computer Society, 2016.
26. J. Hoffstein and J. Silverman. Optimizations for NTRU, January 2000.
27. D. E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.
28. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.
29. Y. Lecun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.
30. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
31. N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Supplementary material

A Complete experimentss with LeNet-5

		Non-proportional perturbation				Proportional perturbation			
		`ReLU`		$\widetilde{\texttt{ReLU}}$		`ReLU`		$\widetilde{\texttt{ReLU}}$	
Pool type	σ	Train acc.	Test acc.	Train acc.	Test acc.	Train acc.	Test acc.	Train acc.	Test acc.
max	0.0	0.9999	0.9924	0.9999	0.9924	0.9999	0.9924	0.9999	0.9924
average		0.9994	0.9903	0.9994	0.9903	0.9994	0.9903	0.9975	0.9903
max	0.1	0.9998	0.9918	0.9996	0.9916	0.9984	0.9908	0.9980	0.9905
average		0.9994	0.9903	0.9993	0.9904	0.9977	0.9891	0.9976	0.9892
max	0.2	0.9990	0.9910	0.9976	0.9899	0.9883	0.9835	0.9842	0.9787
average		0.9991	0.9901	0.9985	0.9894	0.9897	0.9843	0.9878	0.9826
max	0.3	0.9966	0.9894	0.9901	0.9833	0.9540	0.9501	0.9199	0.9192
average		0.9981	0.9898	0.9960	0.9872	0.9699	0.9655	0.9595	0.9581
max	0.4	0.9919	0.9843	0.9654	0.9610	0.8686	0.8723	0.7695	0.7815
average		0.9968	0.9884	0.9908	0.9845	0.9308	0.9296	0.9014	0.9039
max	0.5	0.9823	0.9766	0.8942	0.8966	0.7475	0.7587	0.5901	0.5959
average		0.9947	0.9869	0.9792	0.9737	0.8728	0.8745	0.8156	0.8214
max	0.6	0.9626	0.9610	0.7644	0.7737	0.6199	0.6248	0.4325	0.4317
average		0.9919	0.9842	0.9552	0.9517	0.8007	0.8054	0.7179	0.7245
max	0.7	0.9284	0.9280	0.6166	0.6288	0.5013	0.5024	0.3233	0.3274
average		0.9883	0.9816	0.9171	0.917	0.7219	0.7288	0.6212	0.6332
max	0.8	0.8756	0.8808	0.4809	0.4953	0.4040	0.4056	0.2526	0.2576
average		0.9843	0.9779	0.8633	0.8698	0.6433	0.6506	0.5295	0.5383
max	0.9	0.8103	0.8191	0.3826	0.3884	0.3316	0.3322	0.2036	0.2094
average		0.9779	0.9724	0.8044	0.8135	0.5691	0.5727	0.4498	0.4538
max	1.0	0.7399	0.7462	0.3179	0.326	0.2757	0.2803	0.1719	0.1732
average		0.9696	0.9636	0.7434	0.7548	0.4989	0.5062	0.3822	0.3862

Table 2: Experiments on the LeNet-5 network trained first with max pool and then with average pool. `ReLU` means that during the evaluation the original `ReLU` function was used, while $\widetilde{\texttt{ReLU}}$ signifies that an approximation was used instead.

B Complete experimentss with the Cat and Dog Classifier

		Non-proportional perturbation			Proportional perturbation	
		`ReLU`		$\widetilde{\texttt{ReLU}}$	`ReLU`	$\widetilde{\texttt{ReLU}}$
Pool type	σ	Training acc.	Test acc.	Test acc.	Test acc.	Test acc.
max	0.0	0.9999	0.8530	0.8500	0.8524	0.85
average		0.99995	0.8202	0.8138	0.8232	0.814
max	0.1	0.9944	0.8316	0.8112	0.801	0.784
average		0.99995	0.8232	0.7880	0.8234	0.7812
max	0.2	0.8782	0.7446	0.6246	0.6942	0.5892
average		0.9999	0.8174	0.6726	0.8174	0.6574
max	0.3	0.6234	0.5872	0.5368	0.5736	0.4996
average		0.99965	0.8146	0.5868	0.8134	0.5776
max	0.4	0.5228	0.512	0.5222	0.514	0.4916
average		0.998	0.8074	0.5522	0.8092	0.5444

Table 3: Experiments on a 9-layer CNN trained to distinguish cats from dogs.