Correcting the modulus switch error in TFHE bootstrapping for real-valued computation

Thomas Crasson[1] and Florian Méhats[2]

[1] AMIAD, Pôle Recherche, Palaiseau, France
[2] Sopra Steria Defense & Security, CS Research Lab, Courbevoie, Palaiseau

Abstract. Torus Fully Homomorphic Encryption (TFHE) enables the homomorphic evaluation of arbitrary functions via Programmable Bootstrapping (PBS). However, the modulus switching step inherent to bootstrapping introduces a rounding error that forces the discretization of the input space, limiting the achievable precision on real-valued inputs. We propose a correction algorithm based on a first-order Taylor expansion, applied after bootstrapping, that directly mitigates this rounding error. Our method leverages the many-LUT technique to simultaneously recover encryptions of the function and its derivative within a single PBS, making the correction essentially free in terms of bootstrapping latency. We support our construction with a heuristic average-case noise analysis, validated by empirical measurements, and demonstrate a significant reduction in bootstrapping noise standard deviation. As a proof of concept, we apply our method to the numerical integration of ordinary differential equations under encryption.

Keywords: TFHE · Modulus switch · Numerical Integration · Taylor Expansion

1 Introduction

Fully Homomorphic Encryption (FHE) enables computation over encrypted data without decrypting them. Several schemes have been proposed in order to efficiently perform homomorphic computation. All four schemes: TFHE [CGGI18, CGGI16], BFV [Bra12, FV12], BGV [BGV14] and CKKS [CKKS17], rely on lattice-based cryptography, more specifically on the Learning With Errors (LWE) cryptosystem and its refinement with the Ring Learning With Errors (RLWE) [LPR13, SSTX09]. Each scheme was designed for a specific purpose: BFV and BGV for integer modular arithmetic, TFHE for small integer and boolean operations, and CKKS for approximate real number arithmetic. While CKKS natively supports computations on real-valued numbers, it estimates functions using polynomial expansions, which restricts the class of computable functions. On the other hand, TFHE offers a versatile framework for operating on encrypted data via Programmable Bootstrapping (PBS), enabling the homomorphic evaluation of arbitrary functions – at the cost of requiring the plaintext space to be discretized, which bounds the achievable precision.

In this work, we explore an original positioning: performing approximate homomorphic computation over real numbers, in the spirit of CKKS, but within the TFHE framework. Such a combination is attractive, as it would associate the native real-valued arithmetic of approximate schemes with the ability of TFHE to evaluate arbitrary, possibly non-polynomial, functions through its PBS. However, the PBS does not compute the function but performs a noisy read of a lookup table (LUT) encoded in the test polynomial. The

E-mail: thomas.crasson@polytechnique.edu (Thomas Crasson), florian.mehats@soprasteria.com (Florian Méhats)

input noise, often dominated by the modulus switching error, shifts the read position. The standard TFHE paradigm absorbs this shift by quantizing the plaintext space, so that each message is mapped to a block of identical LUT entries. This redundancy, however, only exists for discrete messages: with real-valued inputs, any shift directly translates into an output error. Our approach therefore departs from the classical use of TFHE, in which the plaintext space consists of small integers or Boolean values and the bootstrapping primarily serves as a noise-reduction step. In our regime, the bootstrapping instead serves to evaluate nonlinear functions on real-valued ciphertexts, and the modulus switching error is an unavoidable approximation cost that we seek to minimize.

We propose to overcome this limitation with a correction algorithm that directly mitigates the modulus switching error. Instead of reducing the rounding error of the modulus switch e_{MS} in the input space of the LUT – as done in prior works – our correction operates in the *output* space: the LUT is evaluated at the perturbed point $x_0 = x + e_{MS}$, and we homomorphically reconstruct $f(x)$ through the first-order Taylor approximation $f(x_0) - e_{MS}\, f'(x_0)$. To compute this correction efficiently, our method leverages the many-LUT technique to simultaneously recover encryptions of the function and its derivative within a single PBS, making the correction essentially free in terms of bootstrapping latency.

Our work has two main limitations. First, it introduces an additional key, the recovery key, which is similar in structure to a PBS key. Because the client must send this key to the server, our technique incurs a higher communication cost than the classical PBS. However, this extra key is optional: the bootstrapping key already encrypts the secret key information required by our correction, and can therefore be directly reused as the recovery key. A dedicated recovery key is only beneficial when the bootstrapping key parameters are chosen aggressively. Second, the modulus switching correction relies on a first-order Taylor expansion, so the evaluated function must be continuously differentiable ($\mathcal{C}^1$). In particular, we do not consider the piecewise-constant LUTs evaluated by the standard PBS. This is a deliberate choice as such LUTs implicitly assume a quantized plaintext space, in which each message is a small integer, whereas our goal is to perform computations on real numbers with TFHE. In this setting, the encrypted values are approximations of continuous quantities, and the relevant functions are themselves continuous. The restriction to $\mathcal{C}^1$ functions is therefore mild in practice: such functions are ubiquitous in scientific computing, for instance in the numerical integration of ODEs, and most activation functions used in modern deep learning, such as GELU, SiLU, sigmoid, tanh and softplus, are smooth.

Related Work

Both the GINX [CGGI18, CGGI16] and AP [LMK+23] bootstrapping algorithms perform a modulus switching from a modulus q to $2N$, where N is the Look-Up Table (LUT) degree. Its rounding noise ultimately forces the quantization of the input space. Previous works have addressed this limitation through several strategies.

A first line of work [CLOT21, BBR26] focuses on making bootstrapping tractable for large LUTs, thereby increasing the number of representable values and refining the input space over which the PBS operates. A second line of work [BCL+25, ML20, KO22, LMP22, BBB+23] encodes large-bit messages into multiple smaller sub-messages and builds efficient arithmetic algorithms on top of these decompositions. While effective, both strategies come at a significant computational cost: the former increases the bootstrapping complexity through larger polynomial degrees, while the latter requires multiple bootstrappings to evaluate a single function.

A third line of work reduces the modulus switching error itself, prior to the blind rotate: the collapse technique [CKLM24, ZYL+18] merges several CMUX steps within the bootstrapping, while the mean compensation [dRDV26] corrects the expectation of the rounding error. In contrast, our algorithm corrects this error *after* the blind rotate

operation.

Recent works have also begun extending PBS-style functional evaluation to CKKS [BCKS24, BKSS24, AKP25], enabling the homomorphic evaluation of non-polynomial functions within the CKKS framework; however, these approaches currently operate on bits or small-integer plaintext spaces, and do not support continuous real-valued inputs.

2 Background

We present in this section the different theoretical elements of the TFHE scheme.

2.1 TFHE for real numbers

TFHE, as other homomorphic schemes, relies on the LWE hardness problem [Reg09] and its generalization RLWE with polynomial ring [LPR13, SSTX09]. We define here the different ciphertext structures used in TFHE as presented in [CGGI18, CGGI16]. In the rest of this work, polynomials will be written with capital letters, while torus elements will be written with small letters.

Definition 1. Let us consider $(q, n, N) \in \mathbb{N}^3$, $\mathbb{T}_N = \mathbb{T}[X]/(X^N + 1)$, where $\mathbb{T}$ is the real torus[1] and let $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$. We define a secret key S as $(S_i)_{i\in\{0,\dots,n-1\}} \in \mathcal{R}_N^n$, with binary or ternary coefficients[2]. The GLWE encryption of a plaintext $x \in \mathbb{T}_N$, denoted $\mathrm{GLWE}_S^{n,N}(x)$, is defined as

$$\mathrm{GLWE}_S^{n,N}(x) = \big(A_0, \cdots, A_{n-1}, B = \sum_{i=0}^{n-1} A_i \cdot S_i + x + e \mod 1\big), \tag{1}$$

where $(A_i)_{i\in\{0,\dots,n-1\}}$ are sampled from $\mathbb{T}_N^n$, $e \in \mathbb{T}_N$ is a Gaussian noise, and $\cdot$ denotes the polynomial ring product. In particular, a LWE_s^n ciphertext can be viewed as a special case where $N = 1$ (in this case, the key is vectorial and thus denoted by small letters) and a RLWE_S^N ciphertext is a special case where $n = 1$.

Note that, in usual TFHE settings, the coefficients of x are taken in a discrete torus of the form $\frac{1}{t}\mathbb{Z}_t$ with $t \ll q$, since TFHE is used as an exact scheme for small integers. Our setting is structurally different and is inspired from approximate schemes like CKKS. Here x is not quantized before encryption, and the noise e induces an irrecoverable approximation error on x, as in CKKS.

Based on the RLWE definition, let us now define the RLev and the RGSW.

Definition 2. Let us consider a setup as in Definition 1, with $n = 1$, and define $(\ell, \beta) \in \mathbb{N}^2$; ℓ is called the level and β is the base. The RLev of a plaintext x is defined as:

$$\mathrm{RLev}_S^{\beta,\ell}(x) = \Big(\mathrm{RLWE}_S^N(x/\beta), \cdots, \mathrm{RLWE}_S^N(x/\beta^\ell)\Big). \tag{2}$$

Definition 3. Considering the same setup as in Definitions 1,2, one defines the RGSW of a plaintext x as:

$$\mathrm{RGSW}_S^{\beta,\ell}(x) = \big(\mathrm{RLev}_S^{\beta,\ell}(-S \cdot x), \mathrm{RLev}_S^{\beta,\ell}(x)\big). \tag{3}$$

[1] In practice, one works in a discrete torus $\frac{1}{q}\mathbb{Z}_q$, but we omit this fact for the simplicity of the presentation.

[2] In this paper, we shall only consider binary keys.

Remark 1 (Security of approximate FHE schemes). Like the CKKS scheme, the present scheme operates in the IND-CPA$^{\mathrm{D}}$ security model rather than standard IND-CPA, due to the approximate nature of decryption [LM21]. This means that an adversary with access to a decryption oracle could in principle exploit the residual noise to recover information about the secret key – the so-called Li-Micciancio attack. Standard mitigations, such as noise flooding prior to decryption, apply directly and restore IND-CPA$^{\mathrm{D}}$ security. Moreover, the primary target applications of the present work – numerical computation and applied mathematics – naturally operate in a compute-only server-side setting, where the server never decrypts and the client decrypts its own results in a trusted environment. In this context, the adversarial decryption oracle assumed by the attack is simply not available.

2.2 Fundamental TFHE Operations

In this section, one recalls a few standard homomorphic operations that will be useful in our construction, and their action on the encryption noise.

2.2.1 Noise estimation model

Throughout this work, we estimate noise growth using a standard average-case approach based on the Central Limit Theorem (CLT), following the methodologies of [CGGI18, BBB+23]. Concretely, we model the coefficients of each noise polynomial as i.i.d. normal random variables, so that noise variances propagate additively and can be tracked via a single scalar at each step. This heuristic model is well-established in the FHE literature and has been shown to give tight noise estimates, see e.g. [CLP20, MP24] where the errors for BGV and BFV schemes are studied. We use it as a supporting tool to dimension our method and complement it with direct empirical measurements over 10000 trials in Section 4 to validate its practical accuracy.

However, we note that this average-case heuristic has recently been called into question, particularly in the context of security-critical noise calibration: [GZ25] discusses several limitations of average-case noise analysis in RLWE-based homomorphic encryption, and [GNSJ24] demonstrates key recovery attacks exploiting non-worst-case noise flooding countermeasures in approximate homomorphic encryption. In our setting, we mainly use the average-case approach to calibrate ρ, defined in Section 3.2 as a numerical precision parameter; security-related parameters, by contrast, should be calibrated using a worst-case analysis.

2.2.2 Homomorphic operations

Ciphertext/Ciphertext Addition Let us recall the addition between two ciphertexts $\mathrm{GLWE}_S^{n,N}(x) = (A_0, ..., A_{n-1}, B)$ and $\mathrm{GLWE}_S^{n,N}(x') = (A'_0, ..., A'_{n-1}, B')$ defined as follows:

$$\mathrm{GLWE}_S^{n,N}(x + x') = (\hat{A}_0, ..., \hat{A}_{n-1}, \hat{B})$$

with $(\hat{A}_i = A_i + A'_i \mod 1)_{i\in\{0,...,n-1\}}$ and $\hat{B} = B + B' \mod 1$. In our noise estimation model, the addition's output noise variance equals the sum of the input noise variances.

Plaintext/Ciphertext multiplication Given a ciphertext $\mathrm{GLWE}_S^{n,N}(x) = (A_0, \ldots, A_{n-1}, B)$ and a plaintext polynomial $P \in \mathcal{R}_N$, the plaintext/ciphertext product is obtained by scaling every component:

$$P \cdot \mathrm{GLWE}_S^{n,N}(x) = \big(P \cdot A_0,\ \ldots,\ P \cdot A_{n-1},\ P \cdot B\big) = \mathrm{GLWE}_S^{n,N}(P \cdot x), \tag{4}$$

The output noise variance equals the input noise variance scaled by $\|P\|_2^2$.

Gadget decomposition. For a base β and a level ℓ, we denote by $\mathrm{Dec}^{\beta,\ell}(a) = \big(a^{(1)}, \dots, a^{(\ell)}\big)$ the signed base-β decomposition of $a \in \mathbb{T}_N$:

$$a \;\approx\; \sum_{j=1}^{\ell} a^{(j)}\, \beta^{-j}, \tag{5}$$

with $a^{(j)} \in \left[-\frac{\beta}{2}, \frac{\beta}{2}\right]$ for all $j \in [1, \ell]$. The error introduced by this approximation follows a uniform law $\mathcal{U}\big(-\beta^{-\ell}/2, \beta^{-\ell}/2\big)$.

External product The external product takes as input a $\mathrm{RLWE}_S^N(P_1) = (A, B)$ and a $\mathrm{RGSW}_S^{\beta,\ell}(P_2) = \big(\mathrm{RLev}_S^{\beta,\ell}(-S \cdot P_2), \mathrm{RLev}_S^{\beta,\ell}(P_2)\big)$, and outputs an $\mathrm{RLWE}_S^N(P_1 \cdot P_2)$, defined as follows:

$$\begin{aligned}\mathrm{RLWE}_S^N(P_1) \boxdot \mathrm{RGSW}_S^{\beta,\ell}(P_2) = &\sum_{j=1}^{\ell} A^{(j)} \cdot \mathrm{RLWE}_S^N(-P_2 \cdot S/\beta^j) \\ &+ \sum_{j=1}^{\ell} B^{(j)} \cdot \mathrm{RLWE}_S^N(P_2/\beta^j),\end{aligned} \tag{6}$$

where $(A^{(j)})_j = \mathrm{Dec}^{\beta,\ell}(A)$ and $(B^{(j)})_j = \mathrm{Dec}^{\beta,\ell}(B)$, and each term is a plaintext/ciphertext multiplication as in (4). Let us define σ^2_{RGSW} (respectively σ^2_{RLWE}) as the noise variance of $\mathrm{RGSW}_S^{\beta,\ell}(P_2)$ (respectively $\mathrm{RLWE}_S^N(P_1)$). The noise variance of the result is equal to

$$\sigma^2_{\mathrm{ext}} = \|P_2\|_2^2 \left(\sigma^2_{RLWE} + \frac{N}{24\beta^{2\ell}}\right) \;+\; \frac{\ell N \beta^2}{6}\, \sigma^2_{RGSW}. \tag{7}$$

Modulus switching The modulus switching operation scales a ciphertext from a modulus 1 to another modulus q' (typically $q' = 2N$, prior to a bootstrap). Given an LWE_S ciphertext $(a_0, \dots, a_{n-1}, b) \in \mathbb{T}^{n+1}$, its modulus switch to q' is

$$\mathrm{ModSwitch}_{q'}(a_0, \dots, a_{n-1}, b) = (\lfloor q' a_0 \rceil, \dots, \lfloor q' a_{n-1} \rceil, \lfloor q' b \rceil) \tag{8}$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. The message is preserved under the same key S. The rounding operation injects an additional rounding noise of variance $n/24$ in the integer domain $\mathbb{Z}_{q'}$, for a binary secret key. Note that this variance is independent of q', as the rounding errors $\lfloor q' a_i \rceil - q' a_i$ are uniform on $[-\frac{1}{2}, \frac{1}{2})$ for any q'.

SampleExtract The SampleExtract operation converts an RLWE ciphertext encrypting a polynomial $P = \sum_{m=0}^{N-1} p_m X^m$ into an LWE ciphertext encrypting a single chosen coefficient p_α, without any homomorphic operation. Let $\mathrm{RLWE}_S^N(P) = (A, B)$ with $A = \sum_m A_m X^m$ and $B = \sum_m B_m X^m$, encrypted under $S = \sum_m s_m X^m$. The extracted key $\bar{s} \in \{0,1\}^N$ is the vector of coefficients of S, namely $\bar{s} = (s_0, \dots, s_{N-1})$. For a target index $\alpha \in \{0, \dots, N-1\}$, $\mathrm{SampleExtract}(\mathrm{RLWE}_S^N(P), \alpha)$ is the $\mathrm{LWE}_{\bar{s}}^N$ ciphertext $(\bar{a}_0, \dots, \bar{a}_{N-1}, \bar{b})$ with $\bar{b} = B_\alpha$ and, for $j \in \{0, \dots, N-1\}$,

$$\bar{a}_j = \begin{cases} A_{\,\alpha - j} & \text{if } 0 \le j \le \alpha, \\ -A_{\,\alpha - j + N} & \text{if } \alpha < j \le N-1. \end{cases} \tag{9}$$

The resulting ciphertext decrypts to p_α under the vectorial key $\bar{s}$, with the same noise as the extracted coefficient and no additional cost.

2.3 Programmable Bootstrapping (PBS)

Starting from the definitions recalled in the previous subsection, we can now define programmable bootstrapping as the operation that takes $\mathrm{LWE}_s^n(x)$ as input and returns a ciphertext $\mathrm{LWE}_{\bar{s}}^N(f(x))$ where f is an arbitrary function. In the rest of this work, we assume that $x \in [-\frac{1}{4}+\delta, \frac{1}{4}-\delta]$ with $\delta > 0$ satisfying $\delta \gg \|e_{init} + e_{MS}\|$, so as to ensure that the bootstrapping output does not overflow the torus in the case of non-periodic functions.

2.3.1 The classical programmable bootstrapping

Let us first present the standard programmable bootstrapping. Usually, PBS consists of four phases: (1) a key switching that moves the input ciphertext from being encrypted under a key of size N – the RLWE polynomial degree – down to a smaller LWE key of dimension n, (2) a modulus switching, (3) a blind rotate that homomorphically rotates a test polynomial, (4) a sample extract.

Our contribution focuses on mitigating the noise introduced during the modulus switching, which is, in most of the cases, the largest noise. Thus, we omit in the rest of this work the initial key switching and consider initially an $\mathrm{LWE}_s^n(x)$ as $(a_0, ..., a_{n-1}, b)$ under a secret key $(s_0, ..., s_{n-1})$. We denote e_{init} as the initial noise of this encryption.

Modulus switching The first step consists in applying the modulus switching operation to the ciphertext input, returning:

$$\tilde{c} = (\tilde{a}, \tilde{b})$$

and where $\tilde{a}$, $\tilde{b}$ are defined as follows:

$$\tilde{a}_i = \lfloor 2Na_i \rceil, \quad \text{for} \quad 0 \leq i \leq n-1, \quad \text{and} \quad \tilde{b} = \lfloor 2Nb \rceil. \tag{10}$$

We shall denote

$$i_0 = \tilde{b} - \sum_i \tilde{a}_i s_i \mod 2N \qquad \text{and} \qquad x_0 = \frac{i_0}{2N}.$$

In this setup, the representative i_0 is chosen in $[-N, N[$. This procedure introduces an error e_{MS} caused by the rounding operations, with

$$e_{MS} = x_0 - x - e_{init} = \frac{1}{2N}\left(\tilde{b} - 2Nb - \sum_{i=0}^{n-1}(\tilde{a_i} - 2Na_i)s_i \mod 2N\right).$$

Notice that, since x has been chosen in an interval of the form $[-1/4+\delta, 1/4-\delta]$, if e_{init} and e_{MS} are small enough, then we have $x_0 \in]-1/4, 1/4[$, thus $i_0 \in]-N/2, N/2[$. Besides, if the a_i's have been sampled uniformly, one can estimate the variance of e_{MS} as:

$$\mathrm{Var}(e_{MS}) = \frac{n}{96N^2}.$$

Blind Rotate We now build the so-called test polynomial $v(x)$ adapted to the target function $x \mapsto f(x)$, by defining its coefficients as:

$$v_i = f\big(\frac{i}{2N}\big) \qquad \text{for } -\frac{N}{2} \leq i \leq \frac{N}{2} - 1.$$

Classically, the test polynomial holds redundancy, repeating several times $f(i)$ in order to mitigate the modulus switch error. However, this redundancy comes with a price. Instead of using efficiently the N values of the test polynomial, classical PBS consumes

several bits to counter the modulus switch noise. In this setup, we do not integrate redundancy in it. We fully quantize the input space $[-\frac{1}{4}, \frac{1}{4}]$ on N values.

The Blind Rotate algorithm consists of computing an RLWE encryption of the following rotated polynomial in $\mathbb{T}_N$:

$$X^{-\tilde{b}+\sum_i \tilde{a}_i s_i}\, v(X) = X^{-i_0}\, v(X).$$

This operation involves n sequential external products, between an accumulator RLWE and the bootstrapping key $(\mathrm{RGSW}_S^{\ell_{BS},\beta_{BS}}(s_i))_{i\in\{0,\dots,n-1\}}$. Thus, considering (7) and similarly to [CLOT21], the output of the blind rotate has an associated noise e_{BS} with a predefined variance σ_{BS}^2, determined as:

$$\sigma_{BS}^2 = nN\left(\frac{\ell_{BS}\beta_{BS}^2\sigma_{BSK}^2}{6} + \frac{1}{48\beta_{BS}^{2\ell}}\right), \tag{11}$$

where σ_{BSK} is the standard deviation of the bootstrapping key.

The Blind Rotate is followed by a SampleExtract. This operation enables to get a LWE ciphertext $(a_0^\star, \cdots, a_{N-1}^\star, b^\star)$ encrypting the coefficient 0 of this polynomial. This operation returns an encryption of message

$$v_{i_0} = f(i_0/(2N)) = f(x_0).$$

To summarize, at the end of this step, we have[3] an $\mathrm{LWE}_{\bar{s}}^N(f(x_0)) = (a_0^\star, \cdots, a_{N-1}^\star, b^\star)$. Decryption process returns the quantity

$$f(x_0) + e_{BS} = b^\star - \sum_{i=0}^{N-1} a_i^\star \bar{s}_i \mod 1$$

instead of $f(x)$. Consequently, the PBS error reads

$$f(x_0) + e_{BS} - f(x) = f(x + e_{MS} + e_{init}) - f(x) + e_{BS}. \tag{12}$$

Therefore, several sources of noise contribute to the difference between the decrypted result and $f(x)$. Among them, the modulus switching error e_{MS} is usually the main source of noise in PBS. For typical cryptographic parameters such as the ones in Subsection 4.1, one computes standard deviations of approximately 1.9×10^{-3} for e_{MS} and 3.5×10^{-8} for e_{BS}. Nevertheless, e_{MS} cannot be easily reduced without increasing the polynomial degree N, at the cost of a proportional increase in bootstrapping complexity. Since e_{MS} acts as an irrecoverable noise on the input value x, this limits the precision of any function evaluation and prevents TFHE from operating reliably on real-valued inputs.

2.3.2 Many-LUT PBS

In this part, we recall the many-LUT bootstrapping. The programmable bootstrapping algorithm described in Part 2.3.1 has been refined in [CLOT21] in order to compute several functions $f_0, ..., f_{2^u-1}$ with $u \in \mathbb{N}^*$. To this end, the modulus switch operation is slightly modified. Let us consider the same input as in Subsection 2.3.1. The modulus switch operation for computing the Many-LUT PBS returns:

$$\tilde{a}_i = 2^u \left\lfloor \frac{2Na_i}{2^u} \right\rceil, \quad \text{for} \quad 0 \le i \le n-1, \quad \text{and} \quad \tilde{b} = 2^u \left\lfloor \frac{2Nb}{2^u} \right\rceil. \tag{13}$$

This new modulus switch induces a new variance for the noise e_{MS} as :

$$\mathrm{Var}(e_{MS}) = \frac{n2^{2u}}{96N^2}. \tag{14}$$

[3] Recall that $\bar{s}$ is the vectorial counterpart of the polynomial key $S = \sum_{i=0}^{N-1} S_i X^i$, i.e. $\bar{s}_i = S_i$ for all i.

In addition, the test polynomial $v(x)$ becomes:

$$v_i = f_j\left(\frac{2^u k}{2N}\right) \quad \text{for } 0 \leq i \leq N-1, \quad k = \left\lfloor \frac{i}{2^u} \right\rfloor, \quad j = i - 2^u k.$$

At the end of the PBS, when performing the SampleExtract, one extracts the first 2^u coefficients of the blind rotate output, each corresponding to $\mathrm{LWE}_{\vec{s}}^N(f_j(x_0))$ for $j \in \{0, \ldots, 2^u - 1\}$. One can observe that this technique introduces a trade-off between the number of evaluated functions 2^u and the added noise e_{MS}. We later use this method with u equal to 1 in order to compute encryption of f and f' with a single bootstrapping.

3 Encrypted Taylor Expansion

This section is devoted to the presentation of our correction algorithm. The starting point is the formula of the PBS error given in (12). For the sake of simplicity, we suppose that $e_{init} \ll e_{MS}$ and omit e_{init} in the rest of this work.

We further assume that the function f is differentiable, which is required for the Taylor expansion underlying our correction technique. Considering the Taylor expansion

$$f(x) = f(x_0) - e_{MS}\, f'(x_0) + \mathcal{O}(e_{MS}^2), \tag{15}$$

our technique, in order to mitigate the modulus switch error, consists in computing an $\mathrm{LWE}_{\vec{s}}^N(e_{MS} f'(x_0))$ and subtracting it from the outcome $\mathrm{LWE}_{\vec{s}}^N(f(x_0))$ of the PBS.

3.1 Overview of the method

In order to homomorphically compute (15), we need to have access to $f(x_0)$ and $f'(x_0)$. Since they are both function outputs of the same input x_0, we can leverage the Many-LUT PBS to get both encrypted messages with a single bootstrapping, avoiding having to compute them with two separate PBS.

Considering (13), we first rewrite e_{MS} according to the many-LUT PBS algorithm with $u = 1$ as

$$e_{MS} = \frac{1}{2N}\left(2\lfloor Nb \rceil - 2Nb - \sum_{i=0}^{n-1}(2\lfloor Na_i \rceil - 2Na_i)s_i \mod 2N\right).$$

Hence, two main challenges arise when computing the Taylor expansion on encrypted data:

1. The terms defining e_{MS} are real-valued fractional numbers in $[-1, 1]$. Since homomorphic multiplication requires integer operands, it is not possible to directly compute the product of e_{MS} with $\mathrm{LWE}_{\vec{s}}^N(f'(x_0))$ in the encrypted domain.

2. Computing the terms $(2\lfloor Na_i \rceil - 2Na_i)s_i$ in the plaintext domain is not possible since we do not have access to secret key values. Note that previous work [dRDV26] estimates this error by replacing each s_i with its expectation $\mathbb{E}(s_i)$. The authors inject this correction term during the modulus switch and before the blind rotate, thereby reducing the e_{MS} by half its variance, $\mathrm{Var}(e_{MS})$, in the case of binary keys.

To tackle these issues, we propose two solutions:

1. We introduce a scaling factor ρ and compute $\lfloor \rho(2\lfloor Nb \rceil - 2Nb) \rceil$ and $\lfloor \rho(2\lfloor Na_i \rceil - 2Na_i) \rceil$. Using these values we can estimate $\lfloor \rho e_{MS} \rceil$. This technique extracts $\log_2(\rho)$ bits of information from ρ. On top of this idea, we encode derivative f' in the LUT test polynomial as $f'/(2N\rho)$ to compute the second term of (15).

2. Although we do not have access to s_i in the plaintext domain, we can provide an RGSW encryption of s_i to compute the sum of $\lfloor \rho(2\lfloor Na_i \rceil - 2Na_i) \rceil s_i$ terms as

$$\sum_{i=0}^{n-1} \lfloor \rho(2\lfloor Na_i \rceil - 2Na_i) \rceil \mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(s_i).$$

In the sequel, the set $(\mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(s_i))_{i\in\{0,\dots,n-1\}}$ are called the recovery key, as it is used to recover the information deleted during the modulus switch. Suitable level ℓ_{RK} and base β_{RK} are chosen according to error analysis.

3.2 Taylor expansion correction

Let us now detail our correction algorithm. To this aim, first rewrite the literal formula of e_{MS} as follows:

$$e_{MS} = \frac{1}{2N}\left(2\lfloor Nb \rceil - 2Nb - \sum_{i=0}^{n-1}(2\lfloor Na_i \rceil - 2Na_i)s_i\right) = \frac{1}{2N}(e_R + e_{ML}) \tag{16}$$

where

$$\begin{aligned} e_R &= \lfloor 2Nb \rceil - 2Nb - \sum_{i=0}^{n-1}(\lfloor 2Na_i \rceil - 2Na_i)s_i, \\ e_{ML} &= 2\lfloor Nb \rceil - \lfloor 2Nb \rceil - \sum_{i=0}^{n-1}(2\lfloor Na_i \rceil - \lfloor 2Na_i \rceil)s_i. \end{aligned} \tag{17}$$

In this computation, we have separated the bias introduced by the rounding operation, corresponding to e_R, from the noise induced by the many-LUT encoding, denoted as e_{ML}. Denote now

$$\begin{aligned} \hat{b}_R &= \lfloor 2Nb \rceil - 2Nb, \qquad \hat{a}_{R,i} = \lfloor 2Na_i \rceil - 2Na_i \qquad \text{for} \quad 0 \le i \le n-1, \\ \hat{b}_{ML} &= 2\lfloor Nb \rceil - \lfloor 2Nb \rceil, \qquad \hat{a}_{ML,i} = 2\lfloor Na_i \rceil - \lfloor 2Na_i \rceil \qquad \text{for} \quad 0 \le i \le n-1. \end{aligned}$$

Remark that e_{ML} is a sum of integer values, whereas e_R is a sum of real-valued numbers in [-1,1]. We thus define

$$e_{\rho R} = \lfloor \rho \hat{b}_R \rceil - \sum_{i=0}^{n-1} \lfloor \rho \hat{a}_{R,i} \rceil s_i.$$

Following the ideas developed in Subsection 3.1, we now introduce our Taylor-based algorithm. It uses the following Many-LUT test polynomial:

$$v(X) = \sum_{i=-N/4}^{N/4-1} \frac{1}{2} f\left(\frac{i}{N}\right) X^{2i} + \frac{1}{2N\rho} f'\left(\frac{i}{N}\right) X^{2i+1}, \tag{18}$$

where negative powers of X are understood modulo $X^N + 1$, i.e. $X^{-k} \equiv -X^{N-k}$ in $\mathcal{R}_N$.

Algorithm 1: Taylor expansion correction

Input: Input ciphertext $\mathrm{LWE}_s^n(x) = (a, b)$
Input: Many-LUT test polynomial $v(X)$ as defined in (18)
Input: Recovery key $(\mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(s_i))_{i\in\{0,\dots,n-1\}}$
Output: Output ciphertext $\mathrm{LWE}_{\bar{s}}^N(f(x))$

1 $(\tilde{a}, \tilde{b}) = \mathrm{ModulusSwitch}(a, b)$
2 $C := \mathrm{RLWE}_S(v(X)X^{-\tilde{b}+\sum_{i=0}^{n-1}\tilde{a}_i s_i}) = \mathrm{BlindRotate}(\tilde{a}, \tilde{b})$ with test polynomial $v(X)$
3 $\mathrm{LWE}_{\bar{s}}^N(f(x_0)) = 2 \times \mathrm{SampleExtract}(C,0)$
4 $E_R = \lfloor \rho \hat{b}_R \rceil$
5 $E_{ML} = \hat{b}_{ML}$
6 **for** $i = 0$ **to** $n-1$ **do**
7 $\quad E_R$ -= $\lfloor \rho \hat{a}_{R,i} \rceil \mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(s_i)$
8 $\quad E_{ML}$ -= $\hat{a}_{ML,i} \mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(s_i)$
9 $C_R = C \boxdot E_R$
10 $C_{ML} = C \boxdot E_{ML}$
11 $\mathrm{LWE}_{\bar{s}}^N(\frac{f'(x_0)}{2N\rho} e_{\rho R}) = \mathrm{SampleExtract}(C_R,1)$
12 $\mathrm{LWE}_{\bar{s}}^N(\frac{f'(x_0)}{2N} e_{ML}) = \mathrm{SampleExtract}(C_{ML},2)$ - $\mathrm{SampleExtract}(C_{ML},0)$
13 $c_{out} = \mathrm{LWE}_{\bar{s}}^N(f(x_0))$ - $\mathrm{LWE}_{\bar{s}}^N(\frac{f'(x_0)}{2N\rho} e_{\rho R})$ - $\mathrm{LWE}_{\bar{s}}^N(\frac{f'(x_0)}{2N} e_{ML})$
14 **return** c_{out}

In the above algorithm, E_R and E_{ML} are initialized as integer scalars and progressively updated into $\mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}$ ciphertexts through the loop, where each plaintext/RGSW operation is understood as a scalar multiple of $\mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(1)$.

In line 2, we perform a many-LUT bootstrapping with two LUTs, the first one encoding the function $f/2$ and the second one encoding $f'/(2N\rho)$. On the one hand, we compute an approximation of $\frac{f'(x_0)}{2N} e_R$ as $\frac{f'(x_0)}{2N\rho} e_{\rho R}$. On the other hand, we approximate $\frac{f'(x_0)}{2N} e_{ML}$ by the forward difference $\frac{f(x_0+\frac{1}{N})-f(x_0)}{2} e_{ML}$, which corresponds to the difference of coefficients 2 and 0 of the rotated test polynomial.

The overhead complexity of this algorithm lies in the for loop and in the computation of C_R and C_{ML}, which involves two external products and n plaintext/ciphertext multiplications. Recalling that the bootstrapping consists of n sequential external products, this computation overhead is negligible compared to bootstrapping latency.

We notice that this algorithm could be combined with the collapse technique proposed in [CKLM24, ZYL+18] to further improve precision. Indeed, the collapse technique modifies the PBS by performing the rounding operation on partial sums, as follows:

$$X^{-\tilde{b}+\sum_{i=0}^{n/m-1}\left\lfloor 2N\sum_{j=0}^{m-1} a_{im+j}\, s_{im+j}\right\rceil} v(X),$$

where m is the collapse parameter. One could then use the scaling factor ρ similarly to our method and compute:

$$\sum_{i=0}^{n/m-1} \sum_{\zeta\in\{0,1\}^m} \lfloor \rho \left(\lfloor 2N\, \alpha_i \cdot \zeta \rceil - 2N\, \alpha_i \cdot \zeta \right) \rceil \mathrm{RGSW}_S^{\beta_{RK},\ell_{RK}}(\mathbf{1}_{\hat{s}_i=\zeta}),$$

where $\alpha_i = (a_{im}, a_{im+1}, \dots, a_{(i+1)m-1})$ and $\hat{s}_i = (s_{im}, s_{im+1}, \dots, s_{(i+1)m-1})$ denote, respectively, the i-th block of m consecutive mask components and the corresponding block of secret key components.

3.3 Noise Analysis

The following Proposition estimates the various parts of the noise after application of our correction algorithm, for a given scaling factor ρ.

Proposition 1 (heuristics)**.** *Consider a smooth function $f : \mathbb{T} \to \mathbb{T}$, a recovery key $(RGSW^{\beta_{RK},\ell_{RK}}(s_i))_{i\in\{0,\dots,n-1\}}$, a scaling factor ρ. Let n be the input secret key size, N be the LUT degree, σ^2_{RK} be the noise variance of the recovery key. In an average case scenario and under the noise model discussed in Subsection 2.3, the output of Algorithm 1 displays a sum of two noises, e_{fhe} and e_{trunc}, that can be estimated as follows.*

(i) The noise e_{fhe} is centered with a variance σ^2_{fhe} equal to:

$$\sigma^2_{fhe} = 4\sigma^2_{BS} + \frac{n}{2}\left(\frac{\rho^2}{12}+1\right)\left(\sigma^2_{BS} + \frac{N}{24\beta_{RK}^{2\ell_{RK}}} + \frac{\ell_{RK} N \beta^2_{RK}\sigma^2_{RK}}{3}\right) + \frac{n}{24}\left(\frac{f'(x)}{2N\rho}\right)^2, \quad (19)$$

where σ_{BS} is defined by (11).

(ii) The noise e_{trunc} is the Taylor expansion truncation error. Its mean value μ_{trunc} and variance σ^2_{trunc} are equal to:

$$\mu_{trunc} = \frac{nf''(x)}{48N^2} \quad (20)$$

$$\sigma^2_{trunc} = \frac{(nf''(x))^2}{1152N^4} \quad (21)$$

Proof. In order to prove Item *(i)*, let us decompose the noise variance σ^2_{fhe} of Algorithm 1's output as

$$\sigma^2_{fhe} = 4\sigma^2_{BS} + \sigma^2_{product} + \sigma^2_{approx}$$

where σ^2_{BS} is the noise variance of the extracted $\mathrm{LWE}^N_{\vec{s}}(f(x_0))$ – the factor 4 arising from the factor 2 applied at line 3 of Algorithm 1 —, $\sigma^2_{product}$ is the noise variance arising from the homomorphic operations (external products) and σ^2_{approx} is the noise variance introduced by the least significant bits of $\hat{b}_R$ and $\hat{a}_{R,i}$ that are discarded when rounding after multiplication by ρ.

Let us estimate separately both variances $\sigma^2_{product}$ and σ^2_{approx}. Denoting

$$v_{out}(X) = v(X)\, X^{-\tilde{b}+\sum_{i=0}^{n-1}\tilde{a}_i s_i},$$

and applying the external product noise formula (7), we obtain

$$\begin{aligned}\sigma^2_{product} &= \mathrm{Var}\Big(\mathrm{RLWE}(v_{out}) \boxdot \mathrm{RGSW}(e_{\rho R})\Big) + 2\,\mathrm{Var}\Big(\mathrm{RLWE}(v_{out}) \boxdot \mathrm{RGSW}(e_{ML})\Big) \\ &= \Big(\mathbb{E}(\|e_{\rho R}\|_2^2) + 2\,\mathbb{E}(\|e_{ML}\|_2^2)\Big)\left(\sigma^2_{BS} + \frac{N}{24\beta_{RK}^{2\ell_{RK}}}\right) \\ &\quad + \frac{\ell_{RK} N \beta^2_{RK}}{6}\Big(\sigma^2_{\mathrm{RGSW}(e_{\rho R})} + 2\,\sigma^2_{\mathrm{RGSW}(e_{ML})}\Big),\end{aligned} \quad (22)$$

where the factor 2 in front of the e_{ML} terms accounts for the subtraction of two independent SampleExtracts at line 12 of Algorithm 1.

Recall that the $\hat{a}_{R,i}$'s are uniform on $[-1/2, 1/2]$, and the $\hat{a}_{ML,i}$'s take values in $\{-1, 0, 1\}$ with probabilities $\{1/4, 1/2, 1/4\}$. Since $e_{\rho R}$ and e_{ML} are linear combinations

of these quantities weighted by the binary secret key (s_i), assumed uniformly distributed over $\{0,1\}$, we obtain

$$\sigma^2_{\mathrm{RGSW}(e_{\rho R})} = \frac{n\rho^2}{12}\sigma^2_{RK}, \qquad \sigma^2_{\mathrm{RGSW}(e_{ML})} = \frac{n}{2}\sigma^2_{RK}. \tag{23}$$

Similarly, since $(s_0, \ldots, s_{n-1})$ is a binary vector, we have

$$\mathbb{E}(\|e_{\rho R}\|_2^2) = \frac{n\rho^2}{24}, \qquad \mathbb{E}(\|e_{ML}\|_2^2) = \frac{n}{4}. \tag{24}$$

Substituting (23) and (24) into (22) gives finally

$$\sigma^2_{product} = \frac{n}{2}\left(\frac{\rho^2}{12}+1\right)\left(\sigma^2_{BS} + \frac{N}{24\beta_{RK}^{2\ell_{RK}}} + \frac{\ell_{RK} N \beta^2_{RK}\sigma^2_{RK}}{3}\right). \tag{25}$$

Let us now estimate σ^2_{approx}, which is the variance of the error arising from the rounding of $\rho\hat{b}_R$ and $\rho\hat{a}_{R,i}$. The approximation error reads

$$e_{approx} = \frac{f'(x_0)}{2N\rho}\left(\rho\hat{b}_R - \lfloor\rho\hat{b}_R\rceil - \sum_{i=0}^{n-1}(\rho\hat{a}_{R,i} - \lfloor\rho\hat{a}_{R,i}\rceil)\, s_i\right). \tag{26}$$

Since $x_0 = x + e_{init} + e_{MS}$ and the errors e_{init}, e_{MS} are small by assumption, one has $f'(x_0) = f'(x) + \mathcal{O}(e_{MS})$. At the order of approximation considered here, we therefore identify $f'(x_0)$ with $f'(x)$. Moreover, $\rho\hat{a}_{R,i} - \lfloor\rho\hat{a}_{R,i}\rceil$ is uniformly distributed on $[-1/2, 1/2]$ and the key is binary. Thus we obtain

$$\sigma^2_{approx} = \left(\frac{f'(x)}{2N\rho}\right)^2 \frac{n}{24}. \tag{27}$$

Summing (25) and (27) yields (19). Item *(i)* of Proposition 1 is proved.

Let us now prove Item *(ii)* of this Proposition. The truncation error e_{trunc} arising from the first-order Taylor expansion is purely numerical and does not involve any homomorphic operation. It is defined by

$$e_{trunc} = \frac{1}{2}f''(x)\, e^2_{MS}.$$

As we perform a Many-LUT PBS with $u = 1$, e_{MS} follows a centered normal distribution with variance given by (14) with $u = 1$, namely $\mathrm{Var}(e_{MS}) = \frac{n}{24N^2}$, and $\mathrm{Var}(e^2_{MS}) = 2\mathrm{Var}(e_{MS})^2$. Therefore, the mean value and variance of e_{trunc}, respectively denoted μ_{trunc} and σ^2_{trunc}, equals to

$$\mu_{trunc} = \frac{nf''(x)}{48N^2}$$

$$\sigma^2_{trunc} = \frac{(nf''(x))^2}{1152N^4}$$

We have proved Item *(ii)* and the proof of Proposition 1 is complete. □

Proposition 1 estimate enables to exhibit an optimal value ρ_{opt} minimizing the variance (19). Indeed, this formula emphasizes a trade-off regarding the value of the scaling factor ρ. On the one hand, increasing ρ tends to decrease the error of the Taylor expansion; on the other hand, the greater ρ is, the more noise we introduce inside the FHE noise. The following result can be directly deduced from Proposition 1 after a straightforward optimisation.

Corollary 1. *Consider the setup defined as in Proposition 1. The optimal value scaling factor minimizing the variance (19) is*

$$\rho_{opt} = \sqrt{\frac{|f'(x)|}{2N}}\left(\sigma_{BS}^2 + \frac{N}{24\beta_{RK}^{2\ell_{RK}}} + \frac{N\ell_{RK}\beta_{RK}^2\sigma_{RK}^2}{3}\right)^{-1/4}. \tag{28}$$

Under this optimal value ρ_{opt}, the variance σ_{fhe}^2 can be written as:

$$\begin{aligned}\sigma_{fhe}^2 = 4\sigma_{BS}^2 + \ & \frac{n}{2}\left(\sigma_{BS}^2 + \frac{N}{24\beta_{RK}^{2\ell_{RK}}} + \frac{\ell_{RK}N\beta_{RK}^2\sigma_{RK}^2}{3}\right) \\ & + \frac{n|f'(x)|}{24N}\left(\sigma_{BS}^2 + \frac{N}{24\beta_{RK}^{2\ell_{RK}}} + \frac{N\ell_{RK}\beta_{RK}^2\sigma_{RK}^2}{3}\right)^{1/2}.\end{aligned} \tag{29}$$

Remark 2. The last term of σ_{fhe}^2 involves $f'(x)$, the derivative of f evaluated at the (unknown) plaintext x. In practice, since f is fixed and known beforehand, one can replace $f'(x)$ by $\|f'\|_2$ to obtain an estimation on σ_{fhe}^2, suitable for an average-case parameter selection. Alternatively, when f' does not vary significantly over the input domain, one may use a representative value such as $f'(0)$ or the average $\mathbb{E}[f'(x)]$ for a more practically oriented estimate.

4 Experiments

In this section, we present three experimental studies. In the first one, we perform general purpose PBS, taking sigmoid and tanh functions as examples as they are standard deep learning activation functions. On top of this, we compare the performance of classical PBS and our correction method for ciphertext/ciphertext multiplication of real-valued numbers. Last, we carry out the numerical integration of a toy one-dimensional ODE. All experiments were conducted with an NVIDIA H100 GPU.

4.1 Homomorphic evaluation of nonlinear functions

In this subsection, we conduct experiments on two typical nonlinear functions, the sigmoid and the tanh functions, with our Algorithm 1. Note that these functions are not, as such, functions from the torus $\mathbb{T}$ to itself: they are defined over the reals. We therefore linearly scale both their inputs and their outputs in order to obtain functions from the torus to the torus. More precisely, let f be one of these functions. We sample x uniformly in $[-1, 1]$ and linearly map it to $[-\frac{1}{4}+\delta, \frac{1}{4}-\delta] \subset \mathbb{T}$ through $t = ax$, with $a = \frac{1}{4} - \delta$; the corresponding output $f(x)$ is likewise linearly mapped into $\mathbb{T}$ through a multiplication by $\frac{1}{4}$. All the noise standard deviations reported in this section are expressed in the real-number scale rather than in the torus scale, therefore the torus errors are multiplied by 4.

The cryptographic parameters are chosen, accordingly with [BBB+23, APS15], as follows:

Table 1: Cryptographic parameters

Parameter	**Value**
λ (Security Level)	128
$\log_2(N)$ (Polynomial size)	11
$\log_2(\sigma_{\text{RLWE}})$	-47
n (LWE size)	1500
$\log_2(\sigma_{\text{LWE}})$	-34
β_{BS} (Bootstrapping base)	2^{11}
ℓ_{BS} (Bootstrapping level)	3
β_{KS} (Keyswitch base)	2^2
ℓ_{KS} (Keyswitch level)	20
β_{RK} (Recovery Key base)	2^8
ℓ_{RK} (Recovery Key level)	8
δ	1.25×10^{-2}

We compare the different noise standard deviations depending on the bootstrapping technique.

Table 2: Noise Standard Deviation for PBS Evaluation of Sigmoid and Tanh

Function	**Exp. Classical PBS**	**Exp. Corrected PBS**	**Theory Corrected PBS**	ρ_{opt}
Sigmoid	1.79×10^{-3}	6.41×10^{-5}	6.40×10^{-5}	41
Tanh	6.17×10^{-3}	1.74×10^{-4}	1.72×10^{-4}	75

Table 2 presents our results for 10000 bootstrappings. We compare the experimental standard deviation of programmable bootstrapping with and without our correction algorithm. One can observe that our correction algorithm significantly reduces the output noise. The third column provides the theoretical standard deviation computed from Proposition 1. Experimental results are very close to the theoretical formula.

On top of this comparison, we performed a latency benchmark between the classical PBS and our corrected PBS. Table 3 presents the latency for the classical PBS and our corrected PBS. As mentioned at the end of Section 3.2, the latency of the correction mechanism is negligible compared to the PBS runtime. The computation overhead induced by the correction is lower than 2% of the PBS latency, which can be considered negligible.

Table 3: Amortized Time Comparison between Classical and Corrected PBS

PBS Type	**Amortized Time (ms)**
Classical PBS	3.92
Corrected PBS	3.99

Comparison with prior works While our method consists in correcting the modulus switch error *after the blind rotate*, we compare our approach against prior works that correct the modulus switch error *before the blind rotate*: the mean compensation technique [dRDV26] and the collapse technique [CKLM24, ZYL+18]. Table 4 shows the noise standard deviation for each technique (the collapse technique is used with the parameter $m = 3$ as in [CKLM24]). It shows that our method is much more efficient.

Table 4: Noise Standard Deviation for three rounding error correction techniques, compared to the classical Programmable Bootstrapping which serves as a reference

Function	**Classical PBS**	**Collapse**	**Mean Compensation**	**Our method**
Sigmoid	1.79×10^{-3}	1.38×10^{-3}	1.27×10^{-3}	6.41×10^{-5}
Tanh	6.17×10^{-3}	4.75×10^{-3}	4.32×10^{-3}	1.74×10^{-4}

4.2 Homomorphic multiplication of real numbers

We present in this section the performance improvements for multiplying real numbers. Usually, in the TFHE setting, LWE/LWE multiplications are not done by relinearization techniques (as for other FHE schemes) but with two parallel PBS, according to the following formula:

$$\mathrm{LWE}(m_1 m_2) = \mathrm{LWE}\left(\left(\frac{m_1+m_2}{2}\right)^2\right) - \mathrm{LWE}\left(\left(\frac{m_1-m_2}{2}\right)^2\right).$$

Table 5: Noise Standard Deviation Comparison for LWE/LWE multiplication

Classical PBS	**Corrected PBS**	ρ_{opt}
8.35×10^{-3}	5.06×10^{-4}	90

Table 5 compares the decryption error for a single LWE/LWE multiplication with the same cryptographic parameters as in Subsection 4.1. These results are consistent with those of Table 2. One can observe that our correction algorithm reduces the noise by roughly a factor of 16. Classical PBS would require a LUT of degree 2^{14} or 2^{15} to achieve similar precision, significantly increasing the computation time. We recall that the computation overhead of our corrected PBS is negligible compared to that of classical PBS, as argued in Section 3.2.

4.3 Numerical Integration of ODEs

In this last subsection, we compute ODE numerical integration with encrypted initial coordinates. Encrypted scientific computing has recently been explored in [KPSL25], where the CKKS scheme is used to solve linear PDEs via finite difference schemes. To the best of our knowledge, the homomorphic numerical integration of *nonlinear* ODEs has not been investigated in previous work. We believe that this use case might have applications in industrial setups, for instance where different parties wish to perform numerical simulations and where one party has the initial coordinates and another has the numerical formula to solve the ODE. Nonetheless, numerical integration requires high precision, motivating the use of our Taylor correction algorithm.

Consider the following toy example ODE:

$$y'(t) = \cos(t) - y(t)^3.$$

We solved it with the second-order Runge–Kutta method (RK2). We performed numerical integration in the plaintext domain for reference, then in the ciphertext domain with and without our correction algorithm. The scheme takes the form

$$y_{n+1} = g_n(y_n)$$

where the function g_n, encoded in a new LUT at each step, equals

$$g_n(y) = y + hf\left(y + \frac{h}{2} f(y, t_n), t_n + \frac{h}{2}\right), \tag{30}$$

with $f(y,t) = \cos(t) - y^3$. It is remarkable that a single PBS is performed at each time step. Since the whole complexity of the scheme is encoded within the LUT, the FHE computation time does not depend on the intricacy of g_n.

We set $x_0 = 0.1$ and sampled 300 unique ciphertexts of x_0. We carried out numerical integration with all ciphertexts. We performed numerical integration with time step $h = 0.025$, for $t \in [0, T]$ with $T = 20$. The optimal ρ value is 15.

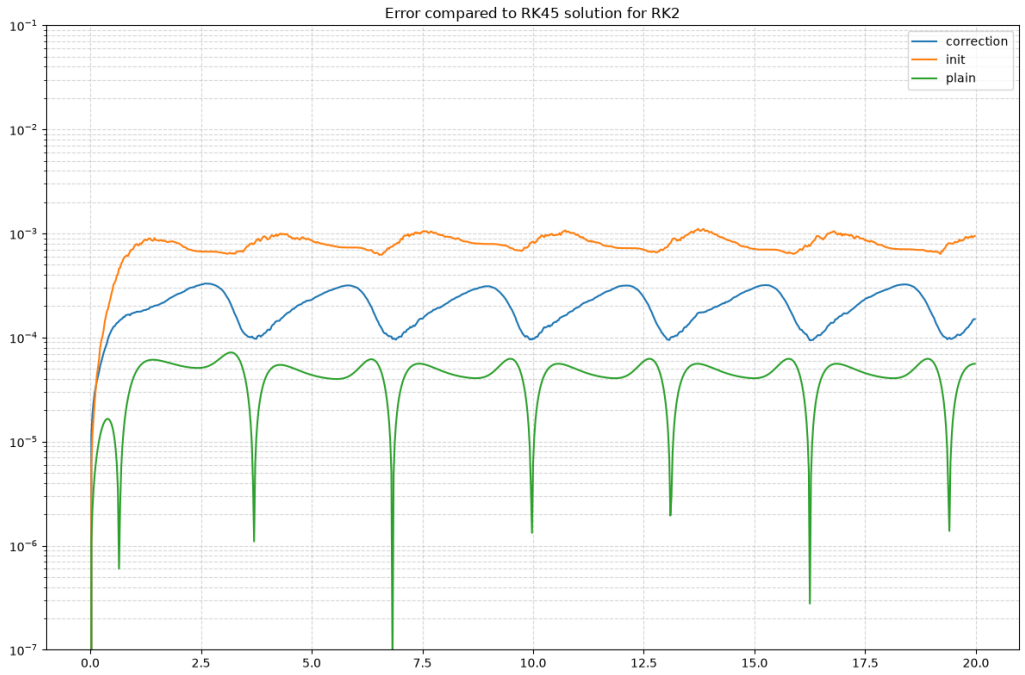


Figure 1: Mean absolute error between the reference solution (obtained with RK45) and the RK2 numerical solutions. The green curve shows the MAE of the solution computed in the plaintext domain, the orange one shows the results for the classical PBS, and the blue one shows the results for our correction algorithm.

We computed a ground-truth solution with the RK45 numerical scheme, and we compare the RK2 integration in three settings: on plaintext data, on ciphertext data with classical PBS, and on ciphertext data with corrected PBS. Figure 1 presents the mean absolute error over time between the RK45 solution and each of these outputs.

RK2 is a second-order method, with a local truncation error in $\mathcal{O}(h^3)$ and a global error in $\mathcal{O}(h^2)$. With our step size, its discretization error is small enough that the error introduced by the modulus switching during PBS becomes dominant in the ciphertext domain. One can observe that our corrected PBS reduces this MAE by a factor of about 4 compared to the classical PBS, which is less significant than in the previous examples. This can be attributed to the accumulation of noise over the $\mathcal{O}(1/h)$ bootstrapping steps, each evaluating a function whose output variation is of order h, so that the per-step signal-to-noise ratio is reduced by a factor of h compared to a single function evaluation.

5 Conclusion

We have presented a Taylor-based correction algorithm that enables homomorphic computation over real numbers within the TFHE framework. By performing a first-order

Taylor expansion to mitigate the modulus switch error, our method achieves an approximately 30-fold reduction in bootstrapping noise with negligible computational overhead, as confirmed by both our theoretical noise analysis and experimental results on sigmoid, tanh, and LWE/LWE multiplication. The agreement between theoretical predictions and empirical measurements validates the relevance of our noise model. As a proof of concept, we have demonstrated that homomorphic numerical integration of ODEs is achievable, a use case that, to the best of our knowledge, has not been previously investigated.

Several directions for future work naturally arise. First, our algorithm can be extended to a second-order Taylor expansion, at the cost of additional parameters, which may further reduce the truncation error for highly irregular functions. Second, the combination of our real-number encoding with multivariate function evaluation opens perspectives toward more complex numerical workflows in the encrypted domain.

This work was performed thanks to grants R1001016831 and AD010617731 from GENCI, IDRIS, and CINES. We thank all three organizations for their support.

References

[AKP25] Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. In *Advances in Cryptology – CRYPTO 2025*, 2025.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[BBB+23] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (T)FHE. *Journal of Cryptology*, 2023.

[BBR26] Sonia Belaïd, Nicolas Bon, and Matthieu Rivain. Decomposition of large look-up tables for fast homomorphic evaluation. Cryptology ePrint Archive, Paper 2026/724, 2026. URL: https://eprint.iacr.org/2026/724.

[BCKS24] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In *Advances in Cryptology – EUROCRYPT 2024*, 2024.

[BCL+25] Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Tfhe gets real: an efficient and flexible homomorphic floating-point arithmetic. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2014.

[BKSS24] Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In *Advances in Cryptology – ASIACRYPT 2024*, 2024.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology – CRYPTO 2012*, 2012.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*, 2016.

[CGGI18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.*, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *LNCS*, pages 409–437. Springer, 2017.

[CKLM24] Philippe Chartier, Michel Koskas, Mohammed Lemou, and Florian Mehats. Fully Homomorphic Encryption on large integers. Cryptology ePrint Archive, Paper 2024/155, 2024. URL: https://eprint.iacr.org/2024/155.pdf.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2021.

[CLP20] Anamaria Costache, Kim Laine, and Rachel Player. Evaluating the effectiveness of heuristic worst-case noise analysis in fhe. In *Computer Security – ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II*, 2020.

[dRDV26] Thomas de Ruijter, Jan-Pieter D'Anvers, and Ingrid Verbauwhede. Don't be mean: Reducing approximation noise in TFHE through mean compensation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(1):82–104, 2026. doi:10.46586/tches.v2026.i1.82-104.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. URL: https://eprint.iacr.org/2012/144.

[GNSJ24] Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with {Non-Worst-Case} noise flooding countermeasures. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7447–7461, 2024.

[GZ25] Mingyu Gao and Hongren Zheng. A critique on average-case noise analysis in rlwe-based homomorphic encryption. In *Proceedings of the 13th Workshop on Encrypted Computing & Applied Homomorphic Computing*, pages 26–37, 2025.

[KO22] Jakub Klemsa and Melek Önen. Parallel operations over tfhe-encrypted multi-digit integers. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, 2022.

[KPSL25] Arseniy Kholod, Yuriy Polyakov, and Michael Schlottke-Lakemper. Secure numerical simulations using fully homomorphic encryption. *Computer Physics Communications*, page 109868, 2025.

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *Advances in Cryptology – EUROCRYPT 2021*, 2021.

[LMK+23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2023*, 2023.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using ffhew/tfhe bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*, 2022.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 2013.

[ML20] Subin Moon and Younho Lee. An efficient encrypted floating-point representation using heaan and tfhe. *Security and Communication Networks*, 2020.

[MP24] Sean Murphy and Rachel Player. A central limit approach for ring-LWE noise analysis. *IACR Communications in Cryptology*, 2024.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *Advances in Cryptology – ASIACRYPT 2009*, 2009.

[ZYL+18] Tanping Zhou, Xiaoyuan Yang, Longfei Liu, Wei Zhang, and Ningbo Li. Faster bootstrapping with multiple addends. *IEEE Access*, 2018.