Fast and Accurate Homomorphic Softmax Evaluation

Wonhee Cho
Department of Mathematics
Seoul National University
Seoul, South Korea
wony404@snu.ac.kr

Guillaume Hanrot*
CryptoLab, Inc.
Lyon, France
guillaume.hanrot@cryptolab.co.kr

Taeseong Kim
Department of Mathematics
Seoul National University
Seoul, South Korea
kts1023@snu.ac.kr

Minje Park
CryptoLab, Inc.
Seoul, South Korea
minje@cryptolab.co.kr

Damien Stehlé
CryptoLab, Inc.
Lyon, France
damien.stehle@cryptolab.co.kr

ABSTRACT

Homomorphic encryption is one of the main solutions for building secure and privacy-preserving solutions for Machine Learning as a Service, a major challenge in a society where AI becomes more and more pervasive. This motivates the development of homomorphic algorithms for the main building blocks of AI, typically for the components of the various types of neural networks architectures.

Among those components, we focus on the Softmax function, defined by $\mathrm{Softmax}(\mathbf{x}) = \left(\exp(x_i) / \sum_{j=1}^{n} \exp(x_j) \right)_{1 \le i \le n}$. This function is deemed to be one of the most difficult to evaluate homomorphically, because of its multivariate nature and of the very large range of values for $\exp(x_i)$. The available homomorphic algorithms remain restricted, especially in large dimensions, while important applications such as Large Language Models (LLM) require computing Softmax over large dimensional vectors. Our algorithm has strong scalability properties in terms of range and dimension while maintaining very good numerical accuracy. In terms of multiplicative depth of the computation (a suitable measure of cost for homomorphic algorithms), our algorithm achieves $O(\log n)$ complexity for a fixed range of inputs, where n is the Softmax dimension.

Our algorithm is especially adapted to the situation where we must compute many Softmax at the same time, for instance, in the LLM situation. In that case, assuming that all Softmax calls are packed into m ciphertexts, the asymptotic amortized multiplicative depth cost per ciphertext is, again over a fixed range, $O(1 + m/N)$ for N the homomorphic ring degree (typically $N = 2^{16}$, so that we have $N \gg m$ in practice).

The main ingredient of our algorithms is a normalize-and-square strategy, which manages to interlace the (numerically unstable) exponential computation over a large range and (very expensive) normalization, decomposing both in stabler and cheaper smaller steps.

We have implemented our algorithms using the HEaaN implementation of the CKKS HE system. Comparing ourselves to the state of the art, our experiments show, in practice, a gain of a factor 2.5 to 8 compared to state of the art solutions.

These experiments demonstrate good accuracy (around 16-bit precision in the worst case, around 20 on average) and support the linear behavior in the dimension. The many-ciphertexts version allows us to compute 8192 Softmax of dimension 256 in parallel

in 486s (single-thread CPU), corresponding to an amortized 0.06s per Softmax call. All Softmax calls of the 32-layers LLaMa large language model (7B version) with context length 128 on an RTX-6000 GPU take around 1.5 minutes, and the final Softmax call in dimension 32768 for token generation takes less than 3 seconds. This suggests that near-practicality may be accessible with dedicated hardware.

1 INTRODUCTION

The fast development of AI technologies in everyday life raises more and more pressing privacy and security concerns. In view of these concerns, Privacy-Preserving Machine Learning has emerged as a major research question over the last few years. We refer to [29] for an overview of the current problems, techniques, and solutions.

In the context of the rapidly expanding "Machine Learning as a Service" (MLaaS) model, Homomorphic Encryption appears to be a promising solution. Initiated by the seminal work of Gentry [14], Homomorphic Encryption (HE) is a cryptographic technique that has since then developed at a fast pace. It allows a server to execute an AI algorithm, typically neural network-based, on a user's encrypted data and produce an encrypted answer while having no information neither on the actual data nor on the answer. Only the user can access the result by decrypting.

On the bright side, homomorphic encryption is very versatile and, from a theoretical point of view, applies "out of the box" to any AI algorithm; on the dark side, despite major progress over the last decade, one of the drawbacks of homomorphic encryption is its high computational cost, making efficient algorithmic design a key to its relevance for a given problem. The deployment of homomorphic encryption for a secure and private MLaaS model thus requires profound algorithmic work on the ML primitives in order to design efficient and adapted solutions.

It is thus of importance to develop HE-tailored algorithms for the most common Machine Learning building blocks, a task which has made a lot of progress over the last few years regarding linear

*Corresponding author.

algebra (see e.g. [17, 20]), convolutional steps (see [19] for a recent reference), and univariate activation functions (see [27] for a recent reference), demonstrating the relevance of HE as a practical solution for privacy-preserving MLaaS.

1.1 Context and Related Work

The implementation of neural networks in the homomorphic encryption context raises several issues. Inference alternates linear steps (linear algebra, convolutions) and non-linear steps (activation functions, normalization steps). In this work, we focus on non-linear steps and, more precisely, on the Softmax step, which probably stands as the most difficult, being multivariate and involving very large or very small intermediate results.

Given that AI algorithms rely on real numbers, the most convenient choice among HE schemes is CKKS[8], which is fully homomorphic (FHE, i.e., allows for arbitrarily complex calculations), and natively enables homomorphic approximate addition, subtraction, and multiplication on real or complex numbers. Additionally, the large inner dimensions of AI primitives makes the fully SIMD mode of operation of CKKS particularly desirable in that setting.

1.1.1 Common strategies for activation functions in HE. Several strategies are possible concerning activation function evaluation. As HE algorithms offer only approximate additions and multiplications (and rotations, to be defined later) as primitives, the starting point is to replace the non-polynomial activation function with a polynomial approximation of it. This requires *a priori* information on the input range of the function, and, depending on the function under study, can be quite a formidable task. For instance, the ReLU function requires [2] a degree $O(2^p)$ polynomial to be evaluated to p bits of precision over $[-1, 1]$. Smoother functions are somewhat less difficult to handle but still give rise to large degree polynomial approximations, which require deep circuits, and hence necessitate "bootstrapping" (see Section 2.2)– an operation which remains, despite a lot of progress, the most expensive HE operation.

Facing this difficulty, we can distinguish two strategies.

The first one consists in designing and using HE-friendly functions, in order to replace the function itself with a simpler adaptation or even a small degree polynomial; sometimes the small degree polynomial is a decent polynomial approximation of the function under study, sometimes it is simply an unrelated small degree polynomial, like in [25] where the x^2 polynomial is used as a replacement to the ReLU function. One drawback of this approach is that it requires retraining to achieve good accuracy for the neural network (if at all possible with the chosen function).

The second one consists in developing HE-efficient algorithms for existing functions, using various strategies in order to try to find the right compromise between efficiency and accuracy; the approximation might be generic, or finely tuned to the properties of the neural network to be evaluated. We refer, for example, to [21, 27].

It remains that, contrary to the first approach, this second setting requires *a priori* knowledge of the range; in between the two approaches, the authors of [30] show that training can be modified to force a sharper range, allowing for a reasonably accurate implementation by a small-degree polynomial. This, however, again requires retraining.

1.1.2 The Softmax function. In this landscape, homomorphically evaluating the Softmax function, defined, for $\mathbf{x} \in \mathbb{R}^n$ by

$$\text{Softmax}(\mathbf{x}) = \left(\frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \right)_{1 \le i \le n}$$

stands as one of the difficult problems to be solved, because of its multidimensional nature, but also due to the fact that its computation, even in plaintext, raises numerical issues which are bound to be even harder in the HE computational model, as the values of $\exp(x_i)$ can be either very large or very small.

The HE implementation of Softmax has been subject to the two families of approaches described above. The first direction was considered in [18], where a sigmoid function is used instead of the Softmax function and as an approximation to it in the multi-class classification problem. In [22], all input values are divided by a constant in order to handle numerical overflows, and a Gumble softmax function is obtained. Finally, in [1], a quadratic polynomial is used as an approximation to Softmax.

In the second direction, the authors of [16] use an iterative "square-and-normalize" approach akin to ours but using an *a priori* normalization, which severely limits its range and accuracy. Finally, the state of the art of this second approach [23] follows the standard non-homomorphic strategy for computing an accurate Softmax, namely first computing and subtracting the maximum value of the input vector to all coordinates; HETAL combines this idea with Domain Extension Polynomials [10] in order to achieve a large input range for the exponential function. However, comparison is a notoriously difficult task in HE and overall implies a deep circuit; the same is also true of the final evaluation of the exp function over a large range.

1.2 Technical contributions

1.2.1 Algorithmic strategy and asymptotic complexity. Let M be a positive real number. For $\mathbf{x} \in [-M, 0]^n$, we use the identity

$$\text{Softmax}(\mathbf{x}/2^{j-1})_i = \frac{\text{Softmax}(\mathbf{x}/2^j)_i^2}{\sum_{t=1}^{n} \text{Softmax}(\mathbf{x}/2^j)_t^2},$$

which follows from the definition, to build an iterative algorithm. If k is an integer such that $k \approx \log M - \log \log n$, we reduce a difficult (and numerically unstable) inversion over $[1/n^{2^k}, 1]$ and a hard evaluation of exp over the large interval $[-M, 0]$ to an evaluation of exp over $[-\log n, 0]$ followed by k squares and easier and stable (inverses or) inverse square-roots over $[1/n, 1]$, using the iteration described in Algorithm 1.

Algorithm 1: Main loop of our Softmax algorithm

$\lambda_j \leftarrow \left(\sum_{i=1}^{n} y_i^{(j-1)^2} \right)^{-1/2}$;

$z_i \leftarrow \lambda_j \cdot y_i^{(j-1)}, i = 1 \dots n$;   /* [Normalization] */

$y_i^{(j)} \leftarrow z_i^2, i = 1 \dots n$;   /* [Squaring] */

When $M < \log n$, our algorithm becomes the naive strategy of evaluating the exponential, then normalizing; we shall thus assume $M \ge \log n$ in the sequel.

For the analysis of our algorithms, we shall use a model that looks simultaneously towards circuit complexity (*multiplicative depth*) and ordinary complexity (*total number of operations*). The *multiplicative depth* is primarily important as it is directly related to the total number of bootstrappings, which are by far the most expensive HE operations. We obtain the following.

THEOREM 1.1. *Let* $M \geq \log n$. *On input* $\mathbf{x} \in [-M, 0]^n$, *Algorithm 2 returns an approximation of* $\mathrm{Softmax}(\mathbf{x})$ *using* $(\log n)(\log M)(1 + o(1))$ *multiplicative levels and* $O((\log M)\sqrt{n \log n})$ *multiplications.*

For fixed M, our level usage has a (quasi[1])linear dependency in $\log n$. We argue that, heuristically, the actual level usage is $(\log n)(\log M)/2(1 + o(1))$. In order to compare ourselves to the state of the art, we provide, in Appendix D, a short analysis of HETAL. This analysis leads to a heuristic multiplicative depth $(\log n)(\log M)(1 + o(1))$.

Our precision analysis bounds the loss of accuracy (compared to the internal precision of CKKS) by $\approx (\log n)(\log M)/2$ bits; however, we provide heuristic arguments that this bound is very pessimistic and that the loss of accuracy should be at most $logM+3(\log n)/2$ bits in the worst case. Our experiments show an even better behaviour than this.

We observe that our algorithm has two threads: a "wide" thread related to the exponential computation that operates on all inputs in parallel and a "thin" thread related to the inverse square root computation, which operates on a single real number at a time. This observation allows us to amortize the simultaneous evaluation of Softmax over a large number of entries using the SIMD capability of CKKS. The following theorem, which focuses on the cost as a function of the number of ciphertexts m, summarizes our results in that case. We assume that the m input ciphertexts encrypt mN_0/n Softmax calls in dimension n, where $m, n \leq N_0$, the number of slots of our HE system. For our complexity measure, we divide each level used by the number of ciphertexts to which it applies; we call the result *amortized level usage.*

THEOREM 1.2. *Let* $m, n \leq N_0$ *two integers. On input* $\mathrm{ct}_1, \ldots, \mathrm{ct}_m$ *containing* mN_0/n *Softmax instances with inputs in* $[-M, 0]$ *in a packed format, the parallel version of our algorithm computes the corresponding Softmax with amortized level usage by ciphertext* $2 \log M + o_m(1)$, *and amortized number of multiplications by ciphertext* $O(\sqrt{\log n} + \log M) + o_m(1)$.

We also describe a variant (see Algorithm 4 in Section 4) which reduces the amortized level usage by a factor of 2, from $\approx 2 \log M$ to $\approx \log M$, up to an increase of the $\log M$ term to $(\log M)^2$ in the number of multiplications.

Compared to this, a many-ciphertext version of HETAL has an amortized level usage by ciphertext of the order of $\approx 4 \log M$, which is 4 times larger than our best solution.

1.2.2 Experimental results. We conclude by an extensive experimental study demonstrating the asymptotic behaviour expected in view of these theoretical analyses for real-world values of n, m, M, with good accuracy (≈ 16 bits).

In particular, in *single thread CPU*, the latency of our algorithms in dimension 128 to 1024 is around 250 seconds, corresponding to

---

[1] recall that $M \geq \log n$

a throughput of the order of 1 Softmax computation per second. If 8192 Softmax of dimension 256 are to be computed simultaneously, we obtain a latency of 414 seconds for a throughput of ≈ 20 Softmax per second.

Finally, we demonstrate the solid scalability properties of our algorithm by the computation of a Softmax of dimension 32768 over [-256, 0]. The computation takes 254 sec, again on a single-thread CPU, with decent accuracy.

Compared to the state-of-the-art solution HETAL [23], we obtain a speedup ranging from 2.5 to 8, the best speedups being obtained in the case where many Softmax need to be computed in parallel.

2 PRELIMINARIES

Notation. We let vectors be denoted in lower-case bold face. For a real number r, we let $\lceil r \rceil$ denote the smallest integer that is no smaller than r. The notation log stands for base-2 logarithm, whereas ln refers to the logarithm in base $e \approx 2.71...$. We let the Hadamard multiplication, namely entry-wise multiplication of vectors, be denoted by $\odot$: formally, given two n-dimensional vectors $\mathbf{v} = (v_1, v_2, \cdots, v_n)$ and $\mathbf{w} = (w_1, w_2, \cdots, w_n)$, $\mathbf{v} \odot \mathbf{w}$ denotes $(v_1 \cdot w_1, v_2 \cdot w_2, \cdots, v_n \cdot w_n)$.

In our analyses, we shall use the $o(\cdot)$ and $O(\cdot)$ Landau notations.

2.1 A quick overview of Homomorphic Encryption

Homomorphic Encryption (HE) is a cryptographic primitive which allows to compute on encrypted data. More precisely, compared to the classical setting of public-key encryption, HE provides one further key, the *evaluation key*, and one further operation, the *evaluation function* Eval, which with the help of the evaluation key is able to evaluate functions from encrypted input and obtain encrypted output; note that the evaluation key does not allow decryption.

In abstract definitions, the evaluation function is assumed to be able to evaluate any circuit. In the descriptions and implementations of concrete schemes, it is most often replaced by a set of specific functions for addition, multiplication, etc.

The HE landscape as of today appears mostly focused around three systems: CGGI/TFHE [12], performing exact computation on booleans or very small integers; BFV/BGV [4, 5, 13], performing exact computations on integers or elements of small finite fields; and CKKS [8], performing approximate computations on real or complex numbers. We shall use the latter one in the sequel.

2.2 The CKKS scheme

The CKKS [8] homomorphic encryption algorithm is a HE cryptosystem that allows one to encode and work on real or complex numbers as messages and to perform approximate arithmetic on the encrypted messages. Given a ring-degree N, CKKS plaintext messages are elements of $\mathbb{C}^{N/2}$, and the ring-homomorphism property of CKKS is related to the ring structure of $(\mathbb{C}^{N/2}, +, \odot)$. The coordinates of a given message are called *slots*.

In the sequel, we shall denote by $N_0 = N/2$ the number of slots of our HE system. The notation n and the word "dimension" will be refer to the dimension of the input vector of the Softmax function, see Eq. (1).

2.2.1 Functionalities. CKKS provides the following functionalities:

- **Key generation**: Given a security parameter λ and a subset $S \subset \{1, \cdots, N_0 - 1\}$, returns a public key pk, a secret key sk, an evaluation key evk, including a set of rotation keys $\{rk_i\}_{i \in S}$.
- **Encryption**: Given a public key pk and a message $\mathbf{m} \in \mathbb{C}^{N_0}$, outputs a ciphertext ct = Enc(pk, $\mathbf{m}$) encrypting $\mathbf{m}$.
- **Decryption**: Given a secret key sk and a ciphertext ct encrypting $\mathbf{m}$, outputs Dec(sk, ct) $\approx m$.
- **Addition**: Given two ciphertexts ct_1 and ct_2 of $\mathbf{m}_1$ and $\mathbf{m}_2$, outputs a ciphertext ct_{add} = Add(ct_1, ct_2) such that we have Dec(sk, ct_{add}) $\approx$ Dec(sk, ct_1) + Dec(sk, ct_2).
- **Multiplication**: Given an evaluation key evk and two ciphertexts ct_1 and ct_2, outputs ct_{mult} = Mult(evk, ct_1, ct_2) such that Dec(sk, ct_{mult}) $\approx$ Dec(sk, ct_1) $\odot$ Dec(sk, ct_2).
- **Rotation**: Given a rotation key rk_i and a ciphertext ct such that Dec(sk, ct) = $(m_0, \cdots, m_{N_0-1})$, outputs a ciphertext Rot(ct, i) with the property that Dec(sk, Rot(ct, i)) = $(m_{i+1}, \cdots, m_{N_0-1}, m_0, \cdots, m_i)$. The rotation index i is defined modulo N_0, namely Rot($\cdot, -i$) means Rot($\cdot, N_0 - i$).

2.2.2 Levels. CKKS comes with a notion of *multiplicative level*. Heuristically speaking, the "quality" of a ciphertext degrades when it goes through multiplications. To formalize this, attached to each ciphertext is a nonnegative integer, its *level*, representing the multiplicative budget left for this ciphertext: if we multiply ct_1 with level l_1 and ct_2 with level l_2, we obtain as a result ct with level $\min(l_1, l_2) - 1$. Once a ciphertext attains a certain bottom level, it must be refreshed by a procedure named *bootstrapping* (BTS) [7] before undergoing further computations. It then recovers a full multiplicative budget.

In practice, one can describe HE algorithms either in a *levelled HE* model, where one considers[2] that the underlying HE parameters allow for a computation of sufficiently large multiplicative depth. However, for large computations a more realistic analysis is obtained by considering the *Fully HE* model, where bootstrapping has to be used at suitable places to restore the level budget. The placement of bootstrapping steps is a question by itself; it is strategic as the bootstrapping steps are, most of the time, by far the most costly steps of the whole computation.

2.2.3 Numerical aspects. The ability of CKKS to natively manipulate complex vectors as messages makes it especially adapted for AI applications. It also means that CKKS only provides one with approximate operations. In practice, the approximate identities above can be rewritten as upper bound estimates on

$$\|\text{Dec(sk, } ct_{add}) - \text{Dec(sk, } ct_1) - \text{Dec(sk, } ct_2)\|_\infty,$$

$$\|\text{Dec(sk, } ct_{mult}) - \text{Dec(sk, } ct_1) \odot \text{Dec(sk, } ct_2)\|_\infty, \text{ etc.,}$$

where $\|(v_i)_{1 \le i \le n}\|_\infty := \max_{1 \le i \le n} |v_i|$ stands for the largest modulus of a coordinate of the vector v. For the corresponding estimates, we refer the reader, e.g., to [8].

In order to reason on CKKS arithmetic and numerical issues, we need to model the behaviour of the underlying arithmetic. Intuitively, CKKS encodes a real number r as $\lfloor r\Delta \rceil$, where Δ is a large

integer, called the scaling factor. It makes the behaviour of CKKS arithmetic very close to that of a *fixed-point system*, implying that from a numerical point of view we can model addition as error-free and other operations introduce an absolute error ε. We shall adopt this error model in the present paper.

2.2.4 Functions as polynomials. The fact that the only arithmetical functions available in an HE context are addition and multiplication implies that the set of functions that one can evaluate is, in essence, restricted to polynomials. A consequence of that fact is that the evaluation of more general functions (we shall require exponential and inverse square root) is performed through the use of *polynomial approximations*. Given a function f over a fixed input interval I and a degree, one finds, using various techniques (Chebyshev interpolation, L^2 approximation, minimax approximation) a polynomial P such that for all $x \in I$, $|P(x) - f(x)| < \varepsilon$. This reduces the evaluation of a function f to the evaluation of the polynomial P.

We shall often use the following facts:

- The evaluation of a polynomial P of degree d can be performed using $\lceil \log(d + 1) \rceil$ multiplicative levels;
- The evaluation of a polynomial P of degree d can be performed using $O(\sqrt{d})$ multiplications between ciphertexts (see [26]).

The cost of evaluating a function is thus directly related to the degree of the polynomial approximation, which itself depends on the range, the "regularity" of the function and the quality requirements for the approximation, namely the value of ε.

2.2.5 Computational model. All these remarks allow us to define a computational model that we shall use to describe and analyze our algorithm:

- Our computation model implements addition, Hadamard multiplication, and rotations on vectors of N_0 slots;
- Numerically speaking, all computations are performed in fixed point arithmetic with p bits of precision following the binary point.
- Our model provides us, via polynomial approximation, with approximations of the functions exp and $x^{-1/2^k}$ for any fixed k.

In Appendix A, we discuss how to build, in theory, those functions from our basic building blocks, with complexity estimates. The practical implementation of those functions is somewhat different from this theoretical study and is discussed in Section 5.

2.3 Complexity model

Since our algorithm targets scalability to situations where many Softmax have to be computed at once in large dimension, we have chosen to provide simultaneously asymptotic estimates (which demonstrate scalability) and implementation results, in order to validate our choice of complexity measures and the asymptotic estimates. For this analysis, we need to define a suitable measure of cost for an HE algorithm.

In short, the folklore wisdom regarding HE is the following:

- Bootstrapping is, by far, the most time-consuming of all operations, even if a lot of work [3, 6, 7, 15, 24], both on the

---

[2]Though feasible in theory, for large computations, this leads to computing with huge HE parameters, which is generally inefficient.

algorithmic and on the implementation sides, has been and is still undertaken to improve its efficiency.

- Apart from this, ciphertext-ciphertext multiplications and rotations are roughly on par and are the most expensive elementary operations.

Orders of magnitude of the cost of those operations for typical CKKS parameters can be found in Section 5.

In view of this, we use level consumption, which is directly linked to the number of bootstrapping calls, as a primary measure of complexity. This is an imperfect model, but in our experience, it gives a fair measure of the efficiency of HE algorithms requiring many levels. We also provide counts for the number of multiplies and rotations, which, combined with the previous, gives a good global picture of the efficiency of our HE algorithms.

3 THE BASELINE SOFTMAX ALGORITHM

In this section, we present and analyze our algorithm for Softmax. For commodity we shall give a non-HE description of our algorithms, not addressing specific HE aspects.

3.1 Definitions, notations, elementary properties

We start by reviewing the definition and useful properties of the Softmax function.

Definition 3.1. Let n be an integer; for $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{R}^n$, we define

$$\text{Softmax}(\mathbf{x}) = \lambda^{-1} \left(\exp(x_1), \ldots, \exp(x_n) \right) \in \mathbb{R}^n, \quad (1)$$

where $\lambda = \sum_{i=1}^n \exp(x_i)$.

We mention two elementary properties of Softmax which are helpful in the discussions to come.

LEMMA 3.2. *For all* $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{R}^n$, *the following identities hold:*

a. *We have, for all* $\mu \in \mathbb{R}$,

$$\text{Softmax}((x_i - \mu)_{1 \leq i \leq n}) = \text{Softmax}(\mathbf{x}); \quad (2)$$

.

b. *If* $\mathbf{y} = \alpha \cdot \text{Softmax}(\mathbf{x})$, $\alpha \neq 0$, *then*

$$\text{Softmax}(\mathbf{x}) = \mathbf{y} / \sum_{i=1}^n y_i; \quad (3)$$

The intuition behind Item b. is that if we have an algorithm that computes Softmax "up to a multiplicative constant," we can recover the actual Softmax thanks to a final normalization step. In other words, we may, to some extent, let errors accumulate as long as they accumulate multiplicatively and in the same way for all coordinates.

3.2 Challenges

We now discuss the numerical challenges associated with computing Softmax, especially in a fixed-point setting. These difficulties will shape our strategy.

3.2.1 Numerical issues. A straightforward implementation of Equation (1) raises difficult issues, even if all inputs are known to lie in $[-M/2, M/2]^n$ for a fixed constant M :

(a) Overflow, if $\exp(M/2)$ is too large to fit within the fixed-point specification;

(b) Symmetrically, underflow if $\exp(-M/2)$ is too small to fit within the fixed-point specification; contrary to overflow, underflow is only an issue in the present context if the largest coordinates of the vector underflow;

(c) In fixed point, any quantity which decreases to $\eta < 1$ loses $-\log \eta$ bits of relative precision, which cannot be recovered if the quantity grows back to its value. Securing precision in such a context requires taking steps to keep data "not too far away from 1" at all times.

(d) In the HE context, the normalization (division by λ) is notoriously difficult, unless one has good control over the interval where λ lives; if we implement Eq. 1 "as is", then we only know that $\lambda \in [n \exp(-M/2), n \exp(M/2)]$ which is typically a huge interval.

In the non-homomorphic setting, one avoids issues (a), (b) by using Eq. (2) with $\mu = \max_{1 \leq i \leq n} x_i$, which ensures that the maximum component of the Softmax vector is equal to 1; it then does not overflow, and the other components can then safely underflow. This also helps significantly with the more HE-specific (d). Item (c) is taken care of automatically by floating-point arithmetic.

HETAL [23] chooses to use the same strategy for (a), (b); however, comparisons are notoriously difficult in HE, and the estimation of M is the most-time consuming step, by far, of their Softmax algorithm. It is then followed by the evaluation of the exponential function over a large interval of width M, which requires domain extension functions [10].

We shall use a (very) poor man's version of this idea, by subtracting a trivial upper bound for $\max_{1 \leq i \leq n} x_i$:

ASSUMPTION 1. *We assume that all our inputs lie within the interval* $[-M, 0]$.

This assumption shall be made in the rest of this paper; we reduce to this situation by using Equation (2). This allows us to avoid issue (a), but not the other ones. We now turn to explain our strategy and how it allows to avoid (b), (c) and limit the impact of (d).

3.2.2 Our strategy. Rather than computing the full vector of exponential values $(\exp(x_i))_{1 \leq i \leq n}$ this way and then divide by the term $\sum_{i=1}^n \exp(x_i)$, we interlace the two computations, using the classical doubling identity $\exp(2x) = \exp(x)^2$ in the following way : if $\mathbf{y} = \text{Softmax}(\mathbf{x})$ or $\mathbf{y} = (\exp(x_i))_{1 \leq i \leq n}$, then, for all i,

$$\text{Softmax}(2\mathbf{x})_i = \frac{y_i^2}{\sum_{j=1}^n y_j^2}. \quad (4)$$

Given a suitable parameter k, we shall compute

$$\mathbf{y}^{(0)} = \left(\exp(x_i/2^k) \right)_{1 \leq i \leq n}$$

directly, then use k times this doubling identity in order to obtain $\mathbf{y}^{(k)} = \text{Softmax}(\mathbf{x})$.

In this way, we trade one inversion for k inversion steps; however, we trade one *hard* inversion for k *much easier* ones. Indeed, the

first number to invert is in $[n \exp(-M/2^k), n]$. For $k \approx \log M$, this is a rather small interval, which makes the inversion tractable. The following lemma considers the following loop iterations.

LEMMA 3.3. *Let* $\mathbf{x} \in \mathbb{R}^n$ *and* $\mathbf{y} = \text{Softmax}(\mathbf{x}/2^j)$. *Then, we have* $\sum_{i=1}^n y_i^2 \in [1/n, 1]$.

PROOF. This follows from $y_i \geq 0$, $\sum_{i=1}^n y_i = 1$ and Cauchy-Schwarz inequality. □

The factor $\lambda^2 := 1/(\sum_{i=1}^n y_i^2)$ is a *normalization factor*. Equation (4) then leads to a "*square-and-normalize*" strategy; for better numerical accuracy, we shall rather adopt a "*normalize-and-square*" strategy, which first computes $z_i = \lambda y_i$, then squares. This is slightly less efficient in computational terms but guarantees that

$$\max z_i \in [n^{-1/2}, 1],$$

at all times. Only then do we compute z_i^2. This allows us to limit the impact of difficulty (c) – a direct implementation of Equation (4) would lead to numbers as small as $1/n^2$ after squaring and before normalization, leading, in fixed point arithmetic, to a potential loss of $2 \log n$ bits of accuracy instead of $\log n$.

REMARK 1. *Thanks to Equation (3), our strategy has a (partial) self-correcting ability: the multiplicative numerical error in the computation of the normalization factors will be corrected during the next normalization step. As the goal of normalization is mostly to "keep quantities close to 1", this allows for a much cheaper approximate normalization. We shall come back to this in Section 5.*

REMARK 2. *Depending on the internal precision and the target precision, the square-and-normalize strategy can be preferred, as in practice, it may lead to a slightly easier computation of the normalization factor (inverse rather than inverse square root). Other normalizations using k-th powers with $k \geq 3$ can be considered; they are less efficient asymptotically but can be of practical interest for small values of n. We discuss this issue in Appendix C.*

3.3 High-level description of the algorithm and precision analysis

We now turn to a formal description of the algorithm. We shall first describe our algorithm in an "ordinary", fixed-point setting, and focus on the precision analysis. We will then explain how to implement this "ordinary" algorithm homomorphically.

Algorithm 2: Softmax

Input: $\mathbf{x} \in [-M, 0]^n, k \in \mathbb{Z}_{>0}$
Output: $\mathbf{y} = \text{Softmax}(\mathbf{x})$.
$y_i^{(0)} \leftarrow \exp(x_i/2^k), i = 1..n$;
for *j from* 1 *to* k **do**
$\quad \lambda_j \leftarrow \left(\sum_{i=1}^n y_i^{(j-1)^2} \right)^{-1/2}$;
$\quad z_i \leftarrow \lambda_j \cdot y_i^{(j-1)}, i = 1..n$;   /* [Normalization] */
$\quad y_i^{(j)} \leftarrow z_i^2, i = 1..n;$ /* [Squaring] */
end
return $(y^{(k)})$

We now give formal statements of (approximate) correctness. Recall that we have fixed the internal precision of our computational model to be p-bit fixed point.

ASSUMPTION 2. *Addition incurs no error in our computational model. We assume that we have native access to an exponential function and an inverse square root function that return a value within 2^{-p} of the actual mathematical value.*

The first part of this assumption is standard in fixed-point models. The second part is discussed in Appendix A. We shall write $\varepsilon = 2^{-p}$.

Note that with exact computations, the correctness of both algorithms follows from the doubling formulas given above. Controlling error amplification leads to a somewhat technical discussion. The reader mostly interested in the practical behaviour of the algorithms should feel free to jump to the implementation section which demonstrates the practical accuracy of Algorithm 2 in typical cases.

THEOREM 3.4. *Let M be a fixed positive real number. We define $k = \lceil \log M - \log \ln n \rceil$ and $\varepsilon = 2^{-p}$ and assume that*

$$\max(n^2, 2.9(2.08\sqrt{n})^k)\varepsilon \leq 1/1000.$$

Algorithm 2, on input $\mathbf{x} \in [-M, 0]^n$, returns a vector $y \in [o(1), 1 + o(1)]^n$ such that

$$\max_{1 \leq i \leq n} |y_i - \text{Softmax}(x)_i| \leq 2.9 \cdot \left((2.08\sqrt{n})^k (n+1) + 15.5n^2 \right) \varepsilon.$$

PROOF. See Appendix B. □

For any $\alpha > 2, \beta > 1$, for n large enough, our proof can be adapted to yield a bound where $2.09(2.08)^k$ is replaced by by $\beta \cdot \alpha^k$. Still, the error estimate is actually extremely pessimistic and far from being observed in practice.

Looking at the proof in a more heuristic way, we are also able to argue in Appendix B that we should rather expect an error increase of $O(n)$ for the normalize-and-square strategy, which, together with the final normalization error (on average) yields a $\approx \log M + 3(\log n)/2$ loss of precision overall.

3.4 Description and cost analysis

In this subsection, we give a description of the homomorphic algorithm; this involves mostly explaining how to compute the values $(\exp(x_i/2^k))_{1 \leq i \leq n}$ at the first step, and then λ_j at each loop iteration $1 \leq j \leq k$, as the other steps can be performed directly using homomorphic addition and multiplication. We refer to Subsection 2.2.4 for a general discussion of function evaluation through polynomial approximation. In this section, we disregard the numerical error issues, in particular in the context of polynomial evaluation.

We recall that we adopt an asymptotic viewpoint and analyze the cost of our approach as a function of n (the dimension) and M (the range). Our cost measures are the ones defined in Section 2.3.

ASSUMPTION 3. *For this analysis, we assume that the number N_0 of slots of our HE algorithm is greater than the Softmax dimension n.*

We shall let the dimension n and the range M tend to infinity independently. We shall allow the precision p to tend to infinity, but no faster than $O(\log n)$ – in view of the typical parameter choices of CKKS this is a realistic assumption for large n.
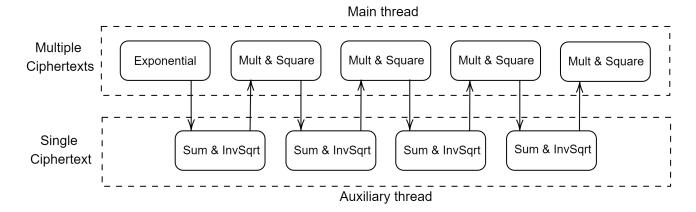
Main thread

Figure 1: High-level Overview of the Softmax Algorithm.

3.4.1 Auxiliary thread and main thread. In order to prepare the description and the analysis of the parallel version, we shall distinguish the computation of the normalization factors λ_j, which we shall call the *auxiliary thread*; we call the rest of the computation the *main thread* (see Figure 1 for an illustration).

It is natural to analyze separately those two parts, mostly because the auxiliary thread is handling essentially one single real number at a time, whereas the main thread operates over n numbers at a time. This distinction prepares the discussion of the case of "many ciphertexts at once" from Section 4.

The full auxiliary thread, except the computation of the inverse square root, is described in Algorithm 3 below. We now analyze it.

LEMMA 3.5. *Let N_0 be the number of slots of our HE implementation, and n an integer dividing N_0. Starting from a ciphertext containing $x_1, \dots, x_n$ in slots $1, 1 + N_0/n, \dots, 1 + (n-1)N_0/n$ and 0 in the remaining slots, we can compute a ciphertext containing $\sum x_i^2$ in slots $1, 1 + N_0/n, \dots, 1 + (n-1)N_0/n$ using one multiplication and $\lceil \log n \rceil$ rotations; if $\lambda = 1/\sqrt{\sum_{i=1}^n x_i^2}$, we can then compute a ciphertext containing x_i/λ in slot $1 + iN_0/n$ using one call to the inverse square root function, two Hadamard products, and $2\lceil \log n \rceil$ rotations.*

PROOF. We use the procedure described in Algorithm 3.   □

Steps 7-10 might seem superfluous, as they deal with a blind spot of our numerical model. In practice, when implementing the algorithm, they are required to achieve a good accuracy in the worst case.

Our numerical model overlooks the fact that in CKKS, the error induced by Steps 3-5 is not the same in all slots, while the self-correcting property of Algorithm 2 is limited to errors which are the same in all slots. After Steps 3-5, we might observe significant variations between slots, which are amplified by the inverse square root computation. Without correction, this makes the overall precision quite poor. Steps 7-10 maintain the same level of error, but average most of the error over the slots – the part of the error which is averaged is then removed by the self-correcting capability of Algorithm 2.

Algorithm 3: Auxiliary thread

> **Input:** $ct_I = (x_1, 0, \dots, x_2, 0, \dots, x_n, 0, \dots, 0)$ containing x_i in slot iN/n
>
> **Output:** $ct_O = (1/\lambda, 0, \dots, 1/\lambda, 0, \dots, 1/\lambda, 0, \dots, 0)$, (nonzero values in slots iN/n), where $\lambda = (\sum_{i=1}^n x_i^2)^{-1/2}$.

1 $ct_O \leftarrow ct_I \odot ct_I$;
2 $mask \leftarrow (1, 0, \dots, 0)$;
3 **for** *j from 0 to* $\lceil \log(n) \rceil - 1$ **do**
4 $\quad ct_O \leftarrow ct_O + \text{Rot}(ct_O, -N/n \cdot 2^j)$
5 **end**
6 $ct_O \leftarrow \text{InvSqrt}(ct_O)$;
7 $ct_O \leftarrow ct_O \odot mask$;
8 **for** *j from 0 to* $\lceil \log(n) \rceil - 1$ **do**
9 $\quad ct_O \leftarrow ct_O + \text{Rot}(ct_O, N/n \cdot 2^j)$
10 **end**
11 **return** ct_O

3.4.2 Asymptotic analysis of the main thread. In levelled HE, the cost of the main thread for Softmax is:

- one evaluation of exp over $[-M/2^k, 0]$;
- two multiplications at each loop iteration.

THEOREM 3.6. *On input $(x_i) \in [-M, 0]^n$, with parameter $k = \log M - \log \ln n + O(1)$, the main thread of Algorithm 2 uses $2 \log M - \log \ln n + O(1)$ levels and $O\left(\sqrt{\log n} + \log M\right)$ ciphertext-ciphertext multiplications.*

PROOF. The loop costs 2 levels per iteration; as $k = \log M - \log \ln n + O(1)$, the estimate follows from the cost of evaluating an exponential over $[-M/2^k, 0] = [-O(\log n), 0]$; see Lemma A.1 in Appendix A.   □

Note that the cost implictly depends on the precision p since we assumed that $p = O(\log n)$.

Table 1 gives the non-asymptotic, practical depth of the computations for our experiments; in the latter, the depth of the main thread did not depend on the Softmax dimension n, only on the

Input range	exp	total depth (main thrd.)
$[-2^k, 0], k \geq 2$	4	$2k + 4$

Table 1: Level usage of the main thread of Algorithm 2 in our experiments

range	dim. n	sq.& mask (Steps 1, 7)	$x^{-1/2}$ (Step 6)	depth (aux. thread)
$[-2^k, 0]$	128	$2^{\times k}$	$6, 4^{\times(k-2)}, 7$	$6k + 5$
$[-2^k, 0]$	256	$2^{\times k}$	$6, 5^{\times(k-2)}, 7$	$7k + 3$
$[-2^k, 0]$	512	$2^{\times k}$	$6, 5^{\times(k-2)}, 7$	$7k + 3$
$[-2^k, 0]$	1024	$2^{\times k}$	$6, 6^{\times(k-2)}, 8$	$8k + 2$

Table 2: Level usage of polynomial approximations and total level usage for the auxiliary thread in our experiments. The meaning of the three figures in the column $x^{-1/2}$ is as follows: the first figure is the depth for loop iteration 1, the notation $d^{\times(k-2)}$ means depth d for the $(k-2)$ intermediate loop iterations, and the last figure is for loop iteration k. Similarly, the notation $d^{\times k}$ in column sq.& mask means depth d for all the k loop iterations.

input range. Our parameters were chosen in order to achieve ≈ 16 bits of accuracy.

3.4.3 Asymptotic analysis of the auxiliary thread. We now turn to the study of the auxiliary thread. We make heavy use of Remark 1: in the description of the algorithm, we allow ourselves to replace the inverse square root by a rough approximation to it (i.e., we replace $1/\sqrt{u}$ by $P(u)$ such that $P(u)^2 \cdot u \in [1/\theta, \theta]$, for θ a constant not too far from 1 – we will write $2^{O(1)}$ instead of θ in the sequel).

In order to estimate the cost of this rough approximation to the inverse square root, we need to make more explicit the input/output intervals of each call to the approximate inverse square root.

- Step 1: the input $\sum_{i=1}^{n} y_i^{(0)^2}$ is in $[n \exp(-M/2^{k-1}), n]$, and we compute an output such that $\lambda_0^2(\sum_{i=1}^{n} y_i^{(0)^2})$ is not too far from 1, or in asymptotic terms $\in [2^{-O(1)}, 2^{O(1)}]$.
- Step $j \in [2, k-1]$: the input is such that $\sum_{i=1}^{n} y_i^{(j)^2} \in [2^{-O(1)}/n, 2^{O(1)}]$ (see Lemma 3.3). Similarly, we expect to have $\lambda_j^2\left(\sum_{i=1}^{n} y_i^{(j)^2}\right) \in [2^{-O(1)}, 2^{O(1)}]$.
- Step $j = k$ is similar, except that we require

$$\lambda_k^2 \left(\sum_{i=1}^{n} y_i^{(k)^2} \right) \in [1 - 2^{-p}, 1 + 2^{-p}].$$

THEOREM 3.7. *For each iteration of the loop, the auxiliary thread of Algorithm 2 requires $\log n(1 + o(1))$ levels and $O(\sqrt{n \log n})$ multiplications.*

PROOF. The result follows from the discussion above and from Lemma A.2 on the polynomial approximation of the inverse square root function. □

As pointed in Appendix A, experimentally, using optimal polynomial approximation, the level consumption can be reduced to $(\log n)/2(1 + o(1))$. This is illustrated by the level usage in Table 2. Similarly, under the same heuristic assumption, the number of ciphertext-ciphertext multiplications should be of the order of $O(n^{1/4})$ rather than $O(\sqrt{n \log n})$.

3.4.4 Practical values of the degrees of the polynomial approximations. Table 2 states the depth of the computation for various dimensions, using the polynomial degrees used in our implementation rather than asymptotic estimations. The total level consumption is the total consumption of the k calls to Algorithm 3.

In practice, the first and the last calls differ from the other ones: for our parameter choices, the first one addresses a somewhat larger interval, while the last one needs to be precise, contrary to the previous ones (see Remark 1).

3.5 Overall evaluation and comparison with HETAL

Using Theorems 3.6 and 3.7, we see that our total level consumption is, in all cases,

$$\leq (2 \log M + (\log M - \log \log n) \log n)(1 + o(1))$$
$$\sim (\log M)(\log n),$$

when n, M tend to infinity.

Again, the discussion concluding Section 3.4.3 suggests that by using optimal polynomial approximations for the inverse square root, this estimate should be divided by 2, i.e. $(\log M) \frac{\log n}{2}$ (see the discussion following Theorem 3.7.

We now compare the algorithm to HETAL [23], the current state of the art for HE-Softmax. We sketch an analysis of the latter in Appendix D; this analysis suggests that HETAL level usage is $\sim (\log n)(\log M)$.

This shows that our algorithm gains a factor of 2 regarding the level consumption. It should however be noted that it comes at the cost of a significant increase in the number of ciphertext-ciphertext multiplications compared to HETAL ($O(n^{1/4} \log M)$ against essentially $O((\log n)(\log M))$). We recall however that level consumption is, asymptotically, a finer measure of cost as it is directly related to the total bootstrapping cost.

4 COMPUTING MANY SOFTMAX AT ONCE

In this section, we discuss the situation where we have a large number L of Softmax on inputs $(x_i^{\{\ell\}})_{1 \leq i \leq n}$, $1 \leq \ell \leq L$, to be computed at the same time.

4.1 The one ciphertext case

If we want to compute in parallel L Softmax of dimension n and $Ln \leq N_0$, the intrinsic SIMD parallelism of CKKS, which can perform simultaneously up to N_0 identical computation on different values, is sufficient.

Indeed, in this case, we may notice that Algorithm 3, on input

$$(x_1^{\{1\}}, x_1^{\{2\}}, \ldots, x_1^{\{L\}}, \mathbf{0}_{N_0/n-L}, x_2^{\{1\}}, x_2^{\{2\}}, \ldots, x_n^{\{L\}}, \mathbf{0}_{N_0/n-L}),$$

where we let $\mathbf{0}_j$ denote a sequence of j zeros, already returns the desired ciphertext containing $x_i^{\{\ell\}}/\sqrt{\sum_{i=1}^n x_i^{\{\ell\}^2}}$ in slot $i + \ell n$.

With this packing, there is thus nothing to modify, and we can take full advantage of the SIMD capability of an HE system like CKKS: the computation time is the same for N_0/n Softmax as it would be for a single one.

4.2 The many ciphertexts case

We now assume that $Ln \geq N_0$, and, for the sake of simplicity, L divides N_0 and that N_0 divides Ln. We can reduce to the previous case by splitting into Ln/N_0 times N_0/n Softmax of size n, but this is sub-optimal, as we only use N_0/n slots of the ciphertext of the auxiliary thread. We let $m = Ln/N_0$ denote the number of ciphertexts of the main thread.

For $L \leq N_0$, one should factor the auxiliary thread computation coming from the various ciphertexts; this means that we pack all the auxiliary threads into one single ciphertext as long as we compute at most N_0 Softmax at once.

Algorithm 3 can be adapted to that setting. Recall our choice of packing:

$$\begin{aligned}
\text{ctxt}_j = (&x_{1+jN_0/L}^{\{1\}}, x_{1+jN_0/L}^{\{2\}}, \ldots, x_{1+jN_0/L}^{\{L\}}, \\
&x_{2+jN_0/L}^{\{1\}}, \ldots, x_{2+jN_0/L}^{\{L\}}, \\
&\ldots, \\
&x_{(j+1)N_0/L}^{\{1\}}, \ldots x_{(j+1)N_0/L}^{\{L\}}), 0 \leq j \leq m-1.
\end{aligned}$$

It allows to have a total number of rotations that is equal to $(m+1)\log(N_0/L) = (m+1)\log(n/m)$ in that case.

A similar idea can be applied to HETAL: as each comparison layer replaces two coordinates in a Softmax by their maximum, the slot usage at each such layer is divided by a factor of 2. The remaining slots can thus be packed into half the number of ciphertexts, until we obtain one single ciphertext.

We introduce the *amortized level consumption per ciphertext* as a suitable measure of the computational cost, as it is directly related to the total number of bootstrapings to be performed, which again dominates the total cost. In order to simplify the statements, we focus on the asymptotic dependency in m, the notation $o_m(1)$ meaning that we neglect lower order terms in m.

THEOREM 4.1. *Assume that $M > \log n$, $N_0 \geq m, n$. On input $\text{ct}_1, \text{ct}_2, \ldots, \text{ct}_m$ packing $L = mN_0/n$ Softmax of dimension n, the total amortized level usage per ciphertext of the many-ciphertext version of Algorithm 2 is*

$$2\log M + \log\log n + O_m(1).$$

The amortized number of ciphertext-ciphertext multiplications per ciphertext is $O(\log M + \sqrt{\log n}) + O_m(1)$.

Our amortized level cost is thus almost constant in $\log n$; in comparison, we estimate the amortized level usage of HETAL to be $\geq 4\log(M) + O_m(1)$, see Appendix D.

4.3 A variant of the main algorithm

We now propose a variant of Algorithm 2, which reduces the level consumption of the main thread per loop iteration from 2 to 1. The

total depth is then $k+1$ plus the depth for exponential evaluation. The main weakness of this approach is the fact that its implementation requires designing one auxiliary function ($x \mapsto x^{-1/2^j}$) for each loop iteration, making it more difficult to implement. It also performs more multiplications overall, which makes it less efficient in the case of a single ciphertext.

Algorithm 4: Softmax (version B)

Input: $\mathbf{x} \in [-M, 0]^n, k \in \mathbb{Z}_{>0}$
Output: $\mathbf{y} = \text{Softmax}(\mathbf{x})$.
$\lambda \leftarrow 1$;
$y_i^{(0)} \leftarrow \exp(x_i/2^k), i = 1..n$;
for j *from* 1 *to* k **do**
$\quad \lambda_j \leftarrow \left(\sum_{i=1}^n y_i^{(j-1)^2}\right)^{-2^{-(j-1)}}$;
$\quad \lambda \leftarrow \lambda \cdot \lambda_j$;
$\quad z_i \leftarrow \lambda \cdot y_i^{(0)}, i = 1..n$;
$\quad y_i^{(j)} \leftarrow z_i^{2^j}, i = 1..n$;
end
return $(y^{(k)})$;

This modification has substantial impact in the many ciphertexts case, as the amortized level consumption of the latter is asymptotically equivalent to the level consumption of the main thread.

From a non-asymptotic point of view, if our range is such that the total level usage fits into the level budget, this approach allows us to totally avoid bootstrapping the main thread, which has a major impact when the number of Softmax to be computed at once (and, thus, the number of ciphertexts) is large. We illustrate this example in Section 5 for $M = 128$.

From an asymptotic point of view, we obtain the following result.

THEOREM 4.2. *With the notations and assumptions of Theorem 4.1, on input $\text{ct}_1, \ldots, \text{ct}_m$, with $N_0 \geq m, n$, the amortized level consumption by ciphertext of the many-ciphertexts version of Algorithm 4 is $\log M + O_m(1)$; the amortized number of ciphertext-ciphertext multiplications is $O(\log^2 M + \sqrt{\log n}) + O_m(1)$.*

This result follows from the fact that the amortized level cost per ciphertext of the main thread is $\log(M/2^k) = \log\log n$ for computing the exponential, and 1 for each of the $k = \log M - \log\log n + O_m(1)$ iterations. The amortized cost per ciphertext of the auxiliary thread is $o_m(1)$ under the assumption that $N_0 \geq m$. For the number of multiplications, the $\sqrt{\log n}$ term comes from the evaluation of exp, while the $O((\log M)^2)$ term comes from the step $y_i^{(j)} \leftarrow z_i^{2^j}$ evaluated for j from 1 to $O(\log M)$.

5 IMPLEMENTATION RESULTS

We have implemented Algorithms 2 and 4. In this section, we describe our implementation and give experimental results supporting our claims regarding efficiency and accuracy.

5.1 Implementation of Algorithms 2 and 4

5.1.1 Software and HE parameters. Our implementation uses the C++ HEaaN library[3]. We use the FGb parameter set, one of the preset HEaaN parameter sets. The corresponding parameters are described in Table 3, together with an estimate for the cost of the homomorphic operations.

This parameter set gives precision (the precision of additions, rotations, multiplications) ≈ 29 bits and bootstrapping precision (the precision of the bootstrapping process) ≈ 22 bits. Overall, 9 multiplicative levels, numbered from 12 to 4, are available before bootstrapping; we can go down to level 0 if we do not need to bootstrap (for instance, upon returning). As already mentioned, bootstrapping restores the full multiplicative budget.

In Table 3, we give timings (in a single-thread CPU context) for the main operations at maximum level $\ell = 12$ (recall that the cost of additions, rotations and multiplications is roughly linear in $\ell + 1$).

Table 3: Overview of the base FGb parameters in the HEaaN library and times for each HE operation. $\log(QP), N_0, L, \lambda$ **denote the bit lengths of the largest modulus, the number of slots (recall that the ring degree N is $2N_0$), the multiplicative depth, and the security parameter. Precision is given in bits.**

$\log(QP)$	N	N_0	L	h	λ	Prec (mult /BTS)
1555	2^{16}	2^{15}	9	192	128	-29 / -22

Time of each homomorphic operation					
enc	dec	add	rot	BTS	mult (pt /ct)
64ms	14ms	2.1ms	130ms	17s	2.6ms / 210ms

5.1.2 Designing auxiliary functions. Several options are possible for implementing the function $x \mapsto x^{-1/2}$, combining minimax approximants and Newton's method; the former choice minimizes the level usage, while the latter reduces the number of multiplications. We have chosen to use minimax approximants, computed using the Sollya [11] tool. In the case of the inverse square root, we use the weighted version of the Remez algorithm, in order to obtain a weighted approximant, namely, a polynomial P which minimizes $\|P(x)\sqrt{x} - 1\|_\infty$ rather than $\|P(x) - 1/\sqrt{x}\|_\infty$. For the exponential function, we have also used minimax approximations. We have chosen degrees of the form $2^t - 1$, as it maximizes accuracy for a fixed-level budget of t.

The implementation of approximate renormalization (see Remark 1) requires some care. First, it should be pointed out that the input of each of the renormalization steps depends on n (but an implementation valid in dimension n will also be valid in smaller dimensions). Using approximate normalization steps means that a given step returns a value λ such that $|\lambda^2 \sum_{i=1}^n y_i^{(j)^2} - 1| \le \alpha$. As a consequence, we have $\sum_{i=1}^n y_i^{(j+1)} \in [1 - \alpha, 1 + \alpha]$ and, using Lemma. 3.3, the input of the next renormalization step is guaranteed to be in $[(1 - \alpha)^2/n, (1 + \alpha)^2]$. Our design thus uses, for Steps 2 to $(k - 1)$, a polynomial such that $|xP(x)^2 - 1| \le \alpha$ over the interval $[(1 - \alpha)^2/n, (1 + \alpha)^2]$.

---

[3]CryptoLab HEaaN library, which is available at https://www.heaan.it

5.1.3 Bootstrapping strategy in the many ciphertexts case. As discussed in Section 4, an efficient implementation of the many ciphertexts case must avoid bootstrapping in the main thread, since this would mean bootstrapping a large number of ciphertexts. In comparison, the bootstrappings of the auxiliary thread are much less expensive, as they can be amortized across the various plaintexts.

In order to achieve our goal of avoiding bootstrappings in the main thread, a careful schedule of the bootstrapping of the auxiliary thread is required, as the output of the auxiliary thread is injected back into the main thread.

Our choice is to minimize the level consumption of the main thread:

- if ct_O has sufficiently high multiplicative budget at Step 6 of Algorithm 3, so as to allow the full evaluation of Algorithm 3 without bootstrapping, we bootstrap the result of Algorithm 3;
- otherwise, we bootstrap ct_O before or after Step 6 of Algorithm 3. If the level of ct_O at the end of Algorithm 3 is lower than the level of the main thread, we bootstrap ct_O again.

Note that the main thread must have one multiplicative level available when entering the auxiliary thread, in order to execute Step 1 of Algorithm 3.

5.1.4 Implementation of Algorithm 4. The purpose of this algorithm is to limit the number of bootstrappings of the main thread. In particular, with our parameters, it is possible to execute Algorithm 4 on the range $[-128, 0]$ without bootstrapping the main thread. We consider Algorithm 4 on this range only.

5.2 Experiments

The goal of our experiments is both to give timings for the interested reader, but also to demonstrate that our asymptotic results are confirmed in practice. We also showcase the performance of Algorithm 4.

It should be noted that our experimental setting, in particular the input range, is derived from our tests on Meta's LLaMa (7B version) LLM [28]. In this setting, the dimension of the Softmax calls correspond to the number of tokens (the sum of the input and output size).

5.2.1 Experimental setting. All our CPU experiments have been run on an Intel Xeon Silver 4114 CPU at 2.2GHz with 260GB of RAM running Linux. All our timings are given for a single-thread execution. We shortly discuss GPU timings at the very end of this section.

5.2.2 Input distribution. We chose to experiment on random inputs. Regarding the distribution of our inputs, we have chosen a normal distribution centered at $-M/2$ with standard deviation $M/6$ (tailcut so that all elements belong to $[-M, 0]$), but found that the uniform distribution on $[-M, 0]$ gives very similar results. Our experiments on LLaMa suggest that this normal distribution is a suitable model for practical inputs to Softmax in this LLM, with $M = 128$. Still, we also experimented using the same distribution for $M = 256$ in order to demonstrate the scalability properties of our method.

5.2.3 Error measurements. As our computations use a fixed-point system, we have mostly measured the absolute error, defined by

$$\text{err}_{\text{abs}} = \|\text{Softmax}(\mathbf{x}) - \text{Algorithm1}(\mathbf{x})\|_{\infty},$$

where for a vector $\mathbf{x}$, the notation $\|\mathbf{x}\|_{\infty}$ stands for the largest coordinate of $\mathbf{x}$ in absolute value. The associated absolute precision is $-\log \text{err}_{\text{abs}}$. This choice seemed more adapted to us due to the fact that our underlying arithmetic is fixed-point.

The alternative would be to measure the *relative precision*, which we define by

$$\text{err}_{\text{rel}} = \frac{\|\text{Softmax}(\mathbf{x}) - \text{Algorithm1}(\mathbf{x})\|_{\infty}}{\|\text{Softmax}(\mathbf{x})\|_{\infty}}.$$

We point that $\|\text{Softmax}(\mathbf{x})\|_{\infty} \geq 1/n$, which shows that the precision difference between the two measures is at most $\log n$ bits. This difference will be large when all values are very close to one another, and small when the maximal value is significantly larger – the second case being more typical than the first one in applications.

In our experiments, we measure the worst-case accuracy but also the average accuracy and standard deviation. The results are obtained via experiments on 5000 Softmax in each dimension. The average accuracy in bits is defined as the logarithm of the average absolute error, and similarly for standard deviation.

5.2.4 The one-ciphertext case. We give in Table 4 the accuracy and time (in single-thread CPU) of Algorithm 2 for computing Softmax in dimension n, in the range $[-256, 0]$.

Table 4: Experimental results for Algorithm 2 for various values of n. All inputs lie within the interval $[-256, 0]$.

M	n	time (s)	precision (bits)		
			worst case	std. dev.	average
	128	124	-15.5	2.0	-21.0
256	256	142	-16.6	1.5	-20.5
	512	145	-15.7	1.0	-19.4
	1024	182	-15.3	0.9	-18.8

We have also tested our algorithm for $n = 32768$ on one input as a stress test. Another motivation is that it is the dimension of the final Softmax call, for the token generation stage of LLaMa. In that case, the computing time was 254s, in line with the linear growth in $\log n$ (we include it on the graph of Figure 2), and the absolute precision was ≈ -12.8 bits in the worst case (-17.2 on average); however, as this absolute precision might be meaningless in that case we measured the relative precision and found it to be ≈ -6.5 bits. It is not surprising that the accuracy degrades somehow. Still, a relative precision of 6.5 bits remain significant enough for many applications, even though more experiments would be needed in order to validate this accuracy.
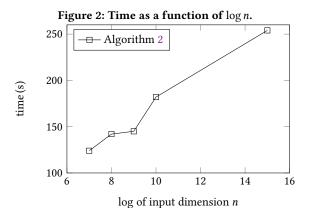
5.2.5 Accuracy. Table 4 demonstrates a very good numerical behaviour of our algorithm. Our precision is limited by the bootstrapping precision (22 bits); the overall loss of precision is thus at most of the order of 6 bits in the worst case, which is significantly better than even our heuristic estimate of $k + 1.5 \log n$ bits.

Further, part of this precision loss occurs by construction. Indeed, our design choices for auxiliary functions reflect our target

precision of 16 bits. In particular, the choice of a 19.5 bit precise approximation for the final square root (see Appendix A) limits, after squaring, the final precision to around 18.5 bits in the worst case.

We have also experimented on Algorithm 4 and found it to have similar accuracy compared to Algorithm 2.

5.2.6 Efficiency. We demonstrate the linear dependency in $\log n$ of the one ciphertext Algorithm 2 in the one ciphertext setting; the timings provided in Table 4, illustrated in Figure 1, support this result. Timings for $n = 512$ are better than expected due to the fact that we are able to use the same degree as for $n = 256$ for the polynomial approximation of the auxiliary thread.

Figure 2: Time as a function of $\log n$.

5.2.7 The many-ciphertext case. In this paragraph, we provide results for the many-ciphertext version of Algorithms 2 and 4 and HETAL.

In order to provide a fair comparison with HETAL in that setting, we have reimplemented the latter using the ideas described in Section 4 to tailor it to the many-ciphertexts case. We provide a comparison based on this implementation.

We have experimented with 1 to 64 simultaneous ciphertexts for Softmax in dimension 256 over [-128, 0]; as each ciphertext packs 128 such Softmax, we shall thus compute up to 8192 Softmax at a time. We point out that these parameter values are typical of current mainstream Large Language Models.

Table 5 gives the timings for the three methods.

Table 5: Comparison between Algorithms 2, 4 and HETAL [23] – times in s., single-thread CPU.

num_ctxt	1	2	4	8	16	32	64
Alg. 2	**130**	**142**	**169**	228	357	617	1080
Alg. 4	160	168	179	**193**	**229**	**286**	**414**
HETAL [23]	388	395	455	605	932	1670	3180
Best speedup	3.0	2.8	2.7	3.1	4.7	5.8	7.7

For 64 ciphertexts in dimension 128, we obtain an amortized cost per Softmax of 0.05s for Algorithm 4, 0.13s for Algorithm 2, and of 0.39s for HETAL's approach.

Finally, we point that for Algorithm 2, the cost of the bootstrappings is ≈ 80% of the total cost; for Algorithm 4, it decreases from ≈ 70% (for 1 ciphertext) to 27% (for 64 ciphertexts). This clearly points to the main difference between the two approaches and illustrates that the key of efficiency lies indeed in the idea of avoiding bootstrappings in the main thread.

5.2.8 GPU implementation. Finally, we have implemented and run a GPU implementation of our results on an NVIDIA RTX-6000 GPU. We have, in that case, simply measured the total cost of all Softmax calls in a full run of LLaMa (7B version) with 128 tokens; in practice, this means 32 times (one per layer) 32 calls to 128 Softmax of dimension 128 (i.e., we are in the many-ciphertext variant with 16 ciphertexts), plus a final call to one Softmax of dimension 32768. The total duration of all these computations was less than 90 seconds. Table 5 shows that the single-thread CPU variant would require around 7600 seconds for the same computation; the GPU architecture thus brings a speedup of a factor ≈ 90.

6 CONCLUSION

We have presented a new HE-compatible Softmax algorithm based on a normalize-and-square strategy. Our experiments demonstrate that it allows real-world computations with GPU timings that come close to practical applicability. Overall, this work will help develop applications of HE to privacy-preserving machine learning.

7 ACKNOWLEDGEMENTS

REFERENCES

[1] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. 2020. PrivFT: Private and Fast Text Classification With Homomorphic Encryption. *IEEE Access* 8 (2020), 226544–226556. https://doi.org/10.1109/ACCESS.2020.3045465

[2] Sergey Bernstein. 1913. Sur la meilleure approximation de |x| par des polynomes de degrés donnés. *Acta. Math.* 37 (1913), 1–57.

[3] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12696)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, 587–617. https://doi.org/10.1007/978-3-030-77870-5_21

[4] Z. Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO*.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory* (2014).

[6] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11477)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, 34–54. https://doi.org/10.1007/978-3-030-17656-3_2

[7] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10820)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, 360–384. https://doi.org/10.1007/978-3-319-78381-9_14

[8] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory*

and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[9] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient Homomorphic Comparison Methods with Optimal Complexity. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12492)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, 221–256. https://doi.org/10.1007/978-3-030-64834-3_8

[10] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. 2022. Efficient Homomorphic Evaluation on Large Intervals. *IEEE Trans. Inf. Forensics Secur.* 17 (2022), 2553–2568. https://doi.org/10.1109/TIFS.2022.3188145

[11] S. Chevillard, M. Joldeş, and C. Lauter. 2010. Sollya: An Environment for the Development of Numerical Codes. In *Mathematical Software - ICMS 2010 (Lecture Notes in Computer Science, Vol. 6327)*, K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama (Eds.). Springer, Heidelberg, Germany, 28–31.

[12] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* 33, 1 (2020), 34–91. https://doi.org/10.1007/S00145-019-09319-X

[13] J. Fan and F. Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Available at http://eprint.iacr.org/2012/144.

[14] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (Bethesda, MD, USA) *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[15] Kyoohyung Han and Dohyeong Ki. 2020. Better Bootstrapping for Approximate Homomorphic Encryption. In *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12006)*, Stanislaw Jarecki (Ed.). Springer, 364–390. https://doi.org/10.1007/978-3-030-40186-3_16

[16] Seungwan Hong, Jai Park, Wonhee Cho, Hyeongmin Choe, and Jung Cheon. 2022. Secure tumor classification by shallow neural network using homomorphic encryption. *BMC Genomics* 23 (04 2022). https://doi.org/10.1186/s12864-022-08469-w

[17] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 1209–1222. https://doi.org/10.1145/3243734.3243837

[18] Chao Jin, Mohamed Ragab, and Khin Mi Mi Aung. 2020. Secure Transfer Learning for Machine Fault Diagnosis Under Different Operating Conditions. In *Provable and Practical Security: 14th International Conference, ProvSec 2020, Singapore, November 29 – December 1, 2020, Proceedings* (Singapore, Singapore). Springer-Verlag, Berlin, Heidelberg, 278–297. https://doi.org/10.1007/978-3-030-62576-4_14

[19] Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Donghwan Kim, and Jung Ho Ahn. 2023. NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and Bootstrapping. *CoRR* abs/2312.04356 (2023). https://doi.org/10.48550/ARXIV.2312.04356 arXiv:2312.04356

[20] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[21] Junghyun Lee, Eunsang Lee, Young-Sik Kim, Yongwoo Lee, Joon-Woo Lee, Yongjune Kim, and Jong-Seon No. 2023. Optimizing Layerwise Polynomial Approximation for Efficient Private Inference on Fully Homomorphic Encryption: A Dynamic Programming Approach. *CoRR* abs/2310.10349 (2023). https://doi.org/10.48550/ARXIV.2310.10349 arXiv:2310.10349

[22] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054. https://doi.org/10.1109/ACCESS.2022.3159694

[23] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. 2023. HETAL: Efficient Privacy-Preserving Transfer Learning with Homomorphic Encryption. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 786, 26 pages.

[24] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. 2022. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13275)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 551–580. https://doi.org/10.1007/978-

3-031-06944-4_19

[25] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy Preserving Low-degree Polynomial Activation for Fast Private Inference. *CoRR* abs/2201.06699 (2022). arXiv:2201.06699 https://arxiv.org/abs/2201.06699

[26] Mike Paterson and Larry J. Stockmeyer. 1973. On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials. *SIAM J. Comput.* 2, 1 (1973), 60–66. https://doi.org/10.1137/0202007

[27] Jianming Tong, Jingtian Dang, Anupam Golder, Callie Hao, Arijit Raychowdhury, and Tushar Krishna. 2024. Accurate Low-Degree Polynomial Approximation of Non-polynomial Operators for Fast Private Inference in Homomorphic Encryption. *CoRR* abs/2404.03216 (2024). https://doi.org/10.48550/ARXIV.2404.03216 arXiv:2404.03216

[28] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[29] Runhua Xu, Nathalie Baracaldo, and James Joshi. 2021. Privacy-Preserving Machine Learning: Methods, Challenges and Directions. arXiv:2108.04417 [cs.LG]

[30] Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. 2023. Converting Transformers to Polynomial Form for Secure Inference Over Homomorphic Encryption. *CoRR* abs/2311.08610 (2023). https://doi.org/10.48550/ARXIV.2311.08610 arXiv:2311.08610

A NON-LINEAR FUNCTION EVALUATION

We recall that we have chosen to focus on mathematical error due to approximation and to neglect the error coming from polynomial evaluation.

A.1 Computing exponentials

LEMMA A.1. *When $A, p \to \infty$, there exists a polynomial P such that $\max_{x \in [-A,0]} |P(x) - \exp(x)| \le 2^{-p}$ and $P(x)$ can be evaluated using*

$$\max(\log p, \log A) + O_{p,A}(1)$$

levels and $O(\max(\sqrt{p}, \sqrt{A})$ ciphertext-ciphertext multiplications.

PROOF. We use the degree d Taylor expansion at 0, whose error (as an alternating series) is $\le A^{d+1}/(d+1)! = 2^{-d \ln(d/(eA))+o(d)}$. For $d = \max(p, e^2 A)$, this is asymptotically less than 2^{-p}.

It then suffices to evaluate a degree d polynomial, which can be performed using $\log d + O(1)$ levels and $O(\sqrt{d})$ ciphertext-ciphertext multiplications. □

A.2 Computing inverse square roots

LEMMA A.2. *For $A, p \to \infty$ and any fixed constant α, there exists a polynomial Q such that $|Q(x) - 1/\sqrt{x}| \le 2^{-p}$ for $x \in [1/(\alpha A), \alpha]$ using $(\log A)(1 + o(1)) + O(\log p)$ levels and $O(\sqrt{A \log A} + \log p)$ ciphertext-ciphertext multiplication.*

PROOF. We consider the case of the interval $[1/A, 1]$, the general case being similar.

We start by computing a truncated Taylor series of $1/\sqrt{x}$ at 1. As the coefficients of this series are ≤ 1 in absolute value, the error of truncating the series at degree k over $[1/A, 1]$ is $O(A(1 - 1/A)^k)$.

For $k = O(A \log A)$ this can be made smaller than any constant; e.g., we compute y_0 such that $|y_0\sqrt{x} - 1| < 1/2$.

We then use Newton's inverse square-root iteration under the form $y_n = y_{n-1} n(3 - x y_{n-1}^2)/2$. This iteration can be evaluated using 2 levels at each step (once $x/2$ is computed) and 4 multiplications using the following algorithm:

$z_1 \leftarrow x/2 \cdot y$;	/* $l(z_1) = \max(l(x/2), l(y)) + 1$ */
$z_2 \leftarrow y \cdot y$;	/* $l(z_2) = l(y) + 1$ */
$z_3 \leftarrow 3/2 \cdot y$;	/* $l(z_3) = l(y) + 1$ */
return $(z_3 - z_1 z_2)$;	/* level $\max(l(x/2), l(y)) + 2$ */

We have

$$|y_n \sqrt{x} - 1| \le |y_{n-1}\sqrt{x} - 1|^2 |y_{n-1}\sqrt{x} + 2|/2.$$

One can check by induction that $y_n\sqrt{x} \le 3/2$, which implies that

$$|y_n\sqrt{x} - 1| \le \frac{7}{4}|y_{n-1}\sqrt{x} - 1|^2 \le \frac{4}{7}\left(\frac{7}{4}|y_0\sqrt{x} - 1|\right)^{2^n} \le \frac{4}{7}(7/8)^{2^n}.$$

Overall, we use a degree $O(A \log A)$ polynomial approximation for computing y_0, so $\log A(1 + o(1))$ levels and $O(\sqrt{A \log A})$ ciphertext-ciphertext multiplications. We then use $O(\log p)$ levels and $O(\log p)$ ciphertext-ciphertext multiplications to then compute $1/\sqrt{x}$ to the precision 2^{-p}. The total number of levels is thus $(\log A)(1 + o(1)) + O(\log p)$, as claimed, and the total number of ciphertext-ciphertext multiplications $O(\sqrt{A \log A} + \log p)$. □

The proof given there should not be used as an efficient algorithm. In practice, one should replace the Taylor expansion of degree $O(A \log A)$ by a minimax approximation of degree $\approx \sqrt{A}/10$ which suffices to bring the Newton iteration within its zone of quadratic convergence. This implies that in practice, the depth of the computation is expected to be $\frac{\log A}{2} \cdot (1 + o(1)) + O(\log p)$, and the number of ciphertext-ciphertext multiplications is expected to be $O(A^{1/4})$.

B PROOF OF THEOREM 3.4

We recall the statement of the theorem.

THEOREM B.1. *Let M be a fixed positive real number. We define $k = \lceil \log M - \log \ln n \rceil$ and $\varepsilon = 2^{-p}$, and assume that*

$$\max(n^2, 2.9(2.08\sqrt{n})^k)\varepsilon \le 1/1000.$$

Algorithm 2, on input $\mathbf{x} \in [-M, 0]^n$, returns a vector $y \in [o(1), 1 + o(1)]^n$ such that

$$\max_{1 \le i \le n} |y_i - \text{Softmax}(x)_i| \le 2.9 \cdot \left((2.08\sqrt{n})^k(n+1) + 15.5n^2\right)\varepsilon.$$

Throughout the analysis, we shall keep the notations of Algorithm 2. When a mathematical quantity bears a tilde symbol, it stands for the numerical approximation computed in the execution of the algorithm; when it does not, it is the exact mathematical value. For example, $\widetilde{\lambda_j} \cdot \widetilde{z_i}$ differs from $\widetilde{\lambda_j \cdot z_i}$ as an error occurs in performing the multiplication.

Before proving the theorem, we prove a lemma which gives estimates on the main quantities used throughout the algorithm; we prove that $\widetilde{\lambda_j}$ is $O(\sqrt{n})$ except maybe at Step 1, and that after the first

step, the vector $(\widetilde{y_i^{(j)}})_{1\le i\le n}$ is always approximately normalized (i.e., has a sum close to 1).

As this lemma will be used inductively, we refrain from using o and O notations, which might imply a hidden exponential growth, and explicit all error terms. Apart from the term $cc'\sqrt{n}$ which plays a role in the final result, we overall chose to favour simplicity over optimality in our estimates.

LEMMA B.2. *Under the assumptions of Theorem B.1, for all j, if $cn \ge \sum_{i=1}^{n} \widetilde{y_i^{(j-1)}} \ge c^{-1}$ with $1 \le c \le 1/(n\varepsilon)$, we have, for all j, the following inequalities:*

$$\widetilde{\lambda_j} \le cc'\sqrt{n} + \varepsilon,$$

$$\left|\sum_{i=1}^{n} \widetilde{y_i^{(j)}} - 1\right| \le 15c^2c'^2n^2\varepsilon,$$

$$\widetilde{z_i} \le 1 + 16c^2c'^2n^2\varepsilon \quad \text{for all } i,$$

for any $c' \ge (1 - c^2n^2\varepsilon)^{-1/2}$.

PROOF. We have $|\widetilde{y_i^{(j-1)^2}} - \widetilde{y_i^{(j-1)}}^2| \le \varepsilon$, so that

$$\left|\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}} - \sum_{i=1}^{n} \widetilde{y_i^{(j-1)}}^2\right| \le n\varepsilon. \tag{5}$$

Our assumption on the inverse square root function implies that

$$\left|\widetilde{\lambda_j} - \left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{-1/2}\right| \le \varepsilon, \tag{6}$$

From $cn \ge \sum_{i=1}^{n} \widetilde{y_i^{(j-1)}} \ge c^{-1}$ we deduce thanks to the Cauchy-Schwarz inequality that $c^2n^2 \ge \sum_{i=1}^{n} \widetilde{y_i^{(j-1)}}^2 \ge c^{-2}/n$. In particular, we have $\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}} \ge c^{-2}/n - n\varepsilon$, so that

$$\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{-1/2} \le c\sqrt{n}(1 - c^2n^2\varepsilon)^{-1/2} \le cc'\sqrt{n}. \tag{7}$$

We also have

$$\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{1/2} \le (c^2n^2 + n\varepsilon)^{1/2} \le 2cn. \tag{8}$$

These yield bounds on $\widetilde{\lambda_j}$ and $\widetilde{\lambda_j}\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{1/2} - 1$ as follows. First, using (6) and (7) we obtain

$$\widetilde{\lambda_j} \le cc'\sqrt{n} + \varepsilon,$$

as claimed. In the end of the proof, we shall mostly use the following weaker bound:

$$\widetilde{\lambda_j} \le 2cc'\sqrt{n}. \tag{9}$$

Then, from (6) and (8), we deduce

$$\left|\widetilde{\lambda_j}\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{1/2} - 1\right| \le \varepsilon\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{1/2} \le 2cn\varepsilon, \tag{10}$$

from which it follows that

$$\left|\widetilde{\lambda_j}^2\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right) - 1\right| \le 2cn\varepsilon\left(1 + \widetilde{\lambda_j}\left(\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}}\right)^{1/2}\right),$$

$$\le 2cn\varepsilon(2 + 2cn\varepsilon),$$

$$\le 8cn\varepsilon. \tag{11}$$

Similarly, from $\left|\widetilde{\lambda_j} \cdot \widetilde{y_i^{(j-1)}} - \widetilde{z_i}\right| \le \varepsilon$, we deduce

$$\left|\widetilde{\lambda_j}^2 \cdot \widetilde{y_i^{(j-1)}}^2 - \widetilde{y_i^{(j)}}\right| \le \left|\left(\widetilde{\lambda_j}\cdot\widetilde{y_i^{(j-1)}} - \widetilde{z_i}\right)\left(\widetilde{\lambda_j}\cdot\widetilde{y_i^{(j-1)}} + \widetilde{z_i}\right)\right|$$

$$+ \left|\widetilde{y_i^{(j)}} - \widetilde{z_i}^2\right|,$$

$$\le \varepsilon(2\widetilde{\lambda_j}\widetilde{y_i^{(j-1)}} + \varepsilon) + \varepsilon$$

$$\le 2\varepsilon\widetilde{\lambda_j}\widetilde{y_i^{(j-1)}} + 2\varepsilon.$$

Summing, we obtain

$$\left|\sum_{i=1}^{n} \widetilde{\lambda_j}^2 \cdot \widetilde{y_i^{(j-1)}}^2 - \sum_{i=1}^{n} \widetilde{y_i^{(j)}}\right| \le 2\varepsilon\widetilde{\lambda_j}\sum_{i=1}^{n} \widetilde{y_i^{(j-1)}} + 2n\varepsilon.$$

From (9) and the assumptions of the lemma, we get

$$\widetilde{\lambda_j}\sum_{i=1}^{n} \widetilde{y_i^{(j-1)}} \le 2c^2c'n^{3/2},$$

from which we deduce

$$\left|\sum_{i=1}^{n} \widetilde{\lambda_j}^2 \cdot \widetilde{y_i^{(j-1)}}^2 - \sum_{i=1}^{n} \widetilde{y_i^{(j)}}\right| \le 4c^2c'n^{3/2}\varepsilon + 2n\varepsilon,$$

$$\le 6c^2c'n^{3/2}\varepsilon. \tag{12}$$

We finally obtain, using (5), (9) and (11),

$$\left|\sum_{i=1}^{n} \widetilde{\lambda_j}^2 \cdot \widetilde{y_i^{(j-1)}}^2 - 1\right| \le \left|\widetilde{\lambda_j}^2\sum_{i=1}^{n} \widetilde{y_i^{(j-1)^2}} - 1\right| + n\widetilde{\lambda_j}^2\varepsilon,$$

$$\le 8cn\varepsilon + c^2c'^2n^2\varepsilon,$$

$$\le 9c^2c'^2n^2\varepsilon,$$

which we combine to (12) to deduce $|\sum_{i=1}^{n} \widetilde{y_i^{(j)}} - 1| \le 15c^2c'^2n^2\varepsilon$, as claimed. As $|\widetilde{y_i^{(j)}} - \widetilde{z_i}^2| \le \varepsilon$ we also have that for all i, $\widetilde{z_i} \le \max(1, \widetilde{z_i}^2) \le 1 + \varepsilon + 15c^2c'^2n^2\varepsilon \le 1 + 16c^2c'^2n^2\varepsilon$, from which the last point follows. □

We now turn to the proof of the theorem.

PROOF. As $k = \lceil \log M - \log \ln n \rceil$, we have $2^k \ge M/\ln n$, so that $x_i \in [-M, 0]$ implies that $x_i/2^k \in [-\ln n, 0]$ for all $1 \le i \le n$. Hence, under our assumptions on the exponential function, $\widetilde{y_i^{(0)}} \in [1/n - \varepsilon, 1 + \varepsilon]$.

We choose n large enough so that $n^2\varepsilon < 1/1000$. Then, the assumptions of the lemma are fulfilled for $j = 1$ as soon as $c \ge (1 - n\varepsilon)^{-1}$; for this it suffices that $c \ge (1 - n^2\varepsilon)^{-1}$. We choose $c = 1.0158$, check that we can choose $c' = 1.0006$, and deduce that

$$1 - 15.5n^2\varepsilon \le \sum_{i=1}^{n} \widetilde{y_i^{(1)}} \le 1 + 15.5n^2\varepsilon.$$

For the following loop iterations, one checks that we can still use $c = 1.0158$ and $c' = 1.0006$, which yield again

$$1 - 15.5n^2\varepsilon \leq \sum_{i=1}^{n} \widetilde{y_i^{(2)}} \leq 1 + 15.5n^2\varepsilon. \tag{13}$$

This shows that the induction carries over and that we can assume that the conclusions of Lemma B.2 apply throughout the algorithm[4] with $c = 1.0158$ and $c' = 1.0006$ With these choices of parameters, the lemma gives, for all i, j,

$$\widetilde{\lambda}_j \leq 1.0158 \cdot 1.0006\sqrt{n} + \varepsilon \leq 1.018\sqrt{n}, \tag{14}$$

$$\widetilde{z_i} \leq 1 + 16.53n^2\varepsilon \leq 1.017. \tag{15}$$

We now estimate the error growth of the normalize-and-square strategy. We prove by induction that for all $j \leq k$, there exists C_j such that

$$|\widetilde{y_i^{(j)}} - C_j \exp(x_i/2^{k-j})| \leq A_j,$$

where A_j is defined, for $j < k$, by $A_0 = \varepsilon$, $A_{j+1} = 2.08\sqrt{n}A_j + 3.036\varepsilon$, so that

$$A_j = (2.08\sqrt{n})^j \varepsilon \left(1 + \frac{3.036}{2.08\sqrt{n}-1}\right) - \frac{3.036}{2.08\sqrt{n}-1}\varepsilon,$$

$$\leq 2.9 \cdot (2.08\sqrt{n})^j \varepsilon. \tag{16}$$

This holds for $j = 0$ with $C_0 = 1$ given our assumptions on the exponential function.

We now fix $j < k$, and assume the induction hypothesis for j. Then, we have, for $1 \leq i \leq n$,

$$|\widetilde{z_i} - \widetilde{\lambda_{j+1}}C_j \exp(x_i/2^{k-j})| \leq |\widetilde{z_i} - \widetilde{\lambda_{j+1}}\widetilde{y_i^{(j)}}| +$$

$$|\widetilde{\lambda_{j+1}}\widetilde{y_i^{(j)}} - \widetilde{\lambda_{j+1}}C_j \exp(x_i/2^{k-j})|,$$

$$\leq \varepsilon + \widetilde{\lambda_{j+1}}A_j, \tag{17}$$

$$\leq \varepsilon + 1.018\sqrt{n}A_j, \tag{18}$$

thanks to (14).

Further, from (14) and (15) we derive

$$|2\widetilde{z_i} + \widetilde{\lambda_{j+1}}A_j + \varepsilon| \leq 2.034 + 1.018\sqrt{n}A_j + 0.001.$$

For $j < k$, Inequality (16) implies $1.018\sqrt{n}A_j \leq 2.9(2.08\sqrt{n})^k\varepsilon < 1/1000$ thanks to our assumption on ε. Overall, we deduce

$$|2\widetilde{z_i} + \widetilde{\lambda_{j+1}}A_j + \varepsilon| \leq 2.036. \tag{19}$$

We have

$$|\widetilde{y_i^{(j+1)}} - \widetilde{z_i}^2| = |\widetilde{z_i^2} - \widetilde{z_i}^2| \leq \varepsilon,$$

---

[4]Note that by making stronger requirements on $n^2\varepsilon$, the constants c and c' can be brought arbitrarily close to 1.

from which we deduce the following chain of inequalities:

$$|\widetilde{y_i^{(j+1)}} - (\widetilde{\lambda_{j+1}}C_j)^2 \exp(x/2^{k-j-1})| \leq$$

$$\varepsilon + |\widetilde{z_i}^2 - (\widetilde{\lambda_{j+1}}C_j)^2 \exp(x/2^{k-j-1})|,$$

$$\leq \varepsilon + |\widetilde{z_i} - (\widetilde{\lambda_{j+1}}C_j)\exp(x/2^{k-j})| \cdot$$

$$|\widetilde{z_i} + (\widetilde{\lambda_{j+1}}C_j)\exp(x/2^{k-j})|,$$

$$\overset{(17)}{\leq} \varepsilon + |\widetilde{z_i} - \widetilde{\lambda_{j+1}}C_j \exp(x_i/2^{k-j})| \cdot$$

$$|2\widetilde{z_i} + \widetilde{\lambda_{j+1}}A_j + \varepsilon|,$$

$$\overset{(18),(19)}{\leq} \varepsilon + 2.036(\varepsilon + 1.018\sqrt{n}A_j)$$

$$\leq 2.08\sqrt{n}A_j + 3.036\varepsilon = A_{j+1}, \tag{20}$$

which concludes the induction by taking $C_{j+1} = (\widetilde{\lambda_{j+1}}C_j)^2$.

We now bound the final normalization error. For $j = k$, we find, for $1 \leq i \leq n$,

$$|\widetilde{y_i^{(k)}} - C_k \exp(x_i)| \leq A_k.$$

Summing, we have

$$\left|C_k - \frac{\sum_{\ell=1}^{n}\widetilde{y_\ell^{(k)}}}{\sum_{\ell=1}^{n}\exp(x_\ell)}\right| \leq \frac{nA_k}{\sum_{\ell=1}^{n}\exp(x_\ell)},$$

so that, for all $1 \leq i \leq n$,

$$\left|\widetilde{y_i^{(k)}} - \sum_{\ell=1}^{n}\widetilde{y_\ell^{(k)}}\mathrm{Softmax}(\mathbf{x})_i\right| \leq A_k + nA_k \frac{\exp(x_i)}{\sum_{\ell=1}^{n}\exp(x_\ell)}$$

$$\leq (n+1)A_k.$$

Finally, using (13), we have

$$\max_{1 \leq i \leq n} |\widetilde{y_i}^{(k)} - \mathrm{Softmax}(\mathbf{x})_i| \leq (n+1)A_k + 15.5n^2\varepsilon,$$

completing the proof. □

REMARK 3. *The final estimate is highly pessimistic. Looking at the proof in a more heuristic way, we can notice that the error induced by the "normalize and square" strategy is expected to be dominated by:*

$$\mathcal{B}_{k,\mathbf{y}^{(0)}} := 2^k \prod_{j=1}^{k} \lambda_j^2 y_i^{(j-1)}, \tag{21}$$

where i is the index for which x_i is maximal (without loss of generality, we assume $i = 1$ in the sequel). By induction, ignoring the various evaluation errors, one checks that, for $j \geq 2$, $1 \leq i \leq n$, the following mathematical identities hold:

$$\lambda_j^2 = \frac{\left(\sum_{\ell=1}^{n} y_\ell^{(0)^{2^{j-1}}}\right)^2}{\sum_{\ell=1}^{n} y_\ell^{(0)^{2^j}}}, \quad y_i^{(j-1)} = \frac{y_i^{(0)^{2^{j-1}}}}{\sum_{\ell=1}^{n} y_\ell^{(0)^{2^{j-1}}}}$$

and that $\lambda_1^2 y_1^{(0)} = y_1^{(0)}/\sum_{\ell=1}^{n} y_\ell^{(0)^2}$, so that

$$\mathcal{B}_{k,\mathbf{y}^{(0)}} = 2^k \frac{y_1^{(0)^{2^k-1}}}{\sum_{\ell=1}^{n} y_\ell^{(0)^{2^k}}} \leq \frac{2^k}{y_1^{(0)}} \leq n2^k,$$

where we have lower bounded the denominator by $y_1^{(0)^{2^k}}$ and the last inequality follows from our choice for k. This heuristic analysis

suggests that in the worst case scenario the normalize and square strategy should lose at most of the order of $k + \log(n)$ bits of precision.

We must also consider the normalization error, related to the error on $\sum_{i=1}^{n} y_i^{(k-1)}$, which on average is expected to be $O(\sqrt{n})$ times the error on $y_i^{(k-1)}$. Overall, we thus expect a total loss of precision of the order of $k + 3(\log n)/2$ bits, which is already a pessimistic estimate compared with our experiments, see Table 4.

C DIFFERENT NORMALIZATION STRATEGIES

For the sake of completeness, we discuss here different normalization strategies. The renormalize-and-square can easily be substituted by a renormalize-and-t-th power strategy. We then use

$$\lambda_j \leftarrow \left(\sum_{i=1}^{n} y_j^{(i)^t} \right)^{1/t}$$

as a renormalization factor.

In practice, the right choice of k goes from $M/2^k \approx \log n$ to $M/t^k \approx \log n$, so we essentially gain a factor $\log t$ on the total depth of the main thread. However, the renormalization step becomes more difficult with larger t. After the first step, t-th power renormalization gives $\sum_{i=1}^{n} y_j^{(i)} = 1$, so by Hölder's inequality we have $\sum_{i=1}^{n} y_j^{(i)^t} \in [n^{-(t-1)}, 1]$.

This range becomes larger with t so that for fixed n, we expect that the number of levels is multiplied by t; overall, we lose a total factor $t/\log t$ in the level consumption. This shows that this strategy may only make sense for a limited range of n, where the asymptotic regime does not reflect the experiments.

D SKETCH OF AN ANALYSIS OF HETAL

We first outline an analysis of the level of consumption of HETAL.

The first stage of the HETAL algorithm uses a hierarchical maximum algorithm; starting with n slots, at the first iteration, it computes an approximation of the maximum of 2 slots, then reduces to compute the maximum of $n/2$ slots. As such, the level cost amounts to $\log n$ hierarchical steps of comparison. In order to achieve approximation within a constant of the final maximum, starting with numbers in $[-M, 0]$, it can be proved that due to the nice properties of the maximum function [9] used by HETAL, it suffices that each pairwise comparison returns a maximum within error $O(1)$ to get an overall maximum within $O(\log n)$. The former is equivalent to evaluating a sign function within error $O(M^{-1})$ on $[-1, 1]$; applying the heuristic result [9, Corollary 5] with "$\alpha = \log M$", we see that each pairwise maximum costs essentially $\log M/\log k$ evaluations of a degree $(2k + 1)$ polynomial, resulting to a level cost of $\log M + O(1)$. The total level cost of the maximum computation is thus $(\log M)(\log n)(1 + o(1))$. The exponential computation and the final normalization are negligible compared to estimating the maximum.

In the many-ciphertext contexts, starting with m ciphertexts, we can group the pairwise maxima in $m/2$ ciphertexts, then $m/4$, etc. The j-th level of the hierarchical maximum tree will thus be executed on $\max(1, m/2^j)$ ciphertexts, for $j \leq \log n$. We overall make a total $\leq \log n + 2m - 1$ calls to the maximum, each for a cost of $\log M(1 + o(1))$, hence an amortized cost of $2 \log M + o_m(1)$ levels per ciphertext. In that case, the exponential computation is no longer negligible; it requires using domain extension polynomials [10]. For the parameters mentioned in [23, Algorithm 1], we find that the level consumption is $2 \log(M/R)/\log L$, for $1.5 < L < 1.5\sqrt{3}$, so $\geq 2.09 \log M(1 + o(1))$ for fixed R. Overall, we find a global cost $\geq 4 \log M(1 + o(1))$ per ciphertext.