Transformer-based Language Models and Homomorphic Encryption: An Intersection with BERT-tiny

Lorenzo Rovida
lorenzo.rovida@unimib.it
University of Milano-Bicocca
Milan, Italy

Alberto Leporati
alberto.leporati@unimib.it
University of Milano-Bicocca
Milan, Italy

ABSTRACT

In recent years, emerging and improved Natural Language Processing (NLP) models, such as Bidirectional Encoder Representations from Transformers (BERT), have gained significant attention due to their performance on several natural language tasks. However, inappropriate focus is usually given to the critical problems of security and data privacy, since these models require access to plain data. To address these issues, we suggest a solution based on Fully Homomorphic Encryption (FHE), which allows for computations to be performed on encrypted data. In particular, we propose a FHE-based circuit that, by implementing the smallest existent BERT model, namely $\text{BERT}_{\text{tiny}}$, enables the extraction of encrypted sentences representations and encrypted text classifications. Considering the nature and the depth of this circuit, we used the Cheon-Kim-Kim-Song (CKKS) scheme, along with the bootstrapping operation. We also propose to use precomputations for the Layer Normalization, in order to lighten computations. The experiments, which can be replicated using our open-source code, are conducted on the Stanford Sentiment Treebank (SST-2) dataset. They show that errors introduced by precomputed Layer Normalizaion, approximate FHE operations and polynomial approximations do not produce a significant performance loss.

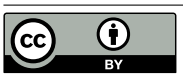
CCS CONCEPTS

• **Computing methodologies** → *Natural language processing*; • **Security and privacy** → **Privacy-preserving protocols**; *Cryptography*.

KEYWORDS

Homomorphic Encryption; Secure Machine Learning; Natural Language Processing

ACM Reference Format:
Lorenzo Rovida and Alberto Leporati. 2024. Transformer-based Language Models and Homomorphic Encryption: An Intersection with BERT-tiny. In *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics (IWSPA '24), June 21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3643651.3659893



IWSPA '24, June 21, 2024, Porto, Portugal

ACM ISBN 979-8-4007-0556-4/24/06
https://doi.org/10.1145/3643651.3659893

1 INTRODUCTION

Bidirectional Encoder Representations from Transformers (BERT) [12] is a revolutionary Natural Language Processing (NLP) model that employs the Transformer [27] architecture. One of the key features of Transformers is the Attention mechanism, which allows to capture contextual information about words and to focus on different relationships among them. Models based on Attention have demonstrated very high performance in many NLP tasks, since they overcome the previous limitations of traditional sequential models. Despite this, insufficient attention is currently given to ensuring data security and privacy.

In this context, Homomorphic Encryption (HE) emerges as a possible solution, as it allows for computations to be performed on encrypted data. The security of almost all known HE schemes is based on the hardness of solving the Learning with Errors (LWE) [24] problem, which is also considered one of the most promising candidates for post-quantum cryptography [3]. Many HE schemes have been proposed in the literature, each having advantages and drawbacks; a good overview is given in [1]. In particular, the Cheon-Kim-Kim-Song (CKKS) scheme [10] well-suits the context of neural networks [11] since it works on complex numbers and has the very beneficial property of batching multiple values in a single ciphertext. It therefore takes advantage of fast Single Instruction, Multiple Data (SIMD) operations, meaning that additions and multiplications are performed slot-wise between encrypted vectors, leading to fast parallel computations. In particular, this work uses the OpenFHE library [2], which offers an error reduced implementation [19] of the Residual Number System (RNS) version of the CKKS scheme.

Since HE adds significant overhead in computations, we propose an implementation of the smallest existing BERT model, called $\text{BERT}_{\text{tiny}}$ [7]. It is a scaled-down version of the BERT model, which consists of $L = 2$ encoders of $H = 128$ hidden units, designed to be more compact and computationally efficient compared to the original BERT. Such efficiency is obtained through the so-called distillation process [26], that trains the smaller model ($\text{BERT}_{\text{tiny}}$) using the outputs or representations of the larger model (BERT) as "teacher" signals.

1.1 Related works

The intersection between NLP and HE schemes has been somewhat explored in the last years. In this section we review the most relevant works that explore this intersection, highlighting the differences with respect to our approach. The main issue, currently unsolved, is the overhead introduced by HE computations, which prevents Large Language Models (LLM) to be implemented.

In 2020, [6] proposed PrivFT, perhaps the first work that showed practical text classification using the *fasttext* [17] architecture and

the first RNS variant for CKKS accelerated by GPU. The implementation showed very good performance. Nevertheless, the *fasttext* architecture is much simpler than Transformers, and the evaluation of softmax function is performed by the client.

A couple of years later, [9] introduced THE-X, a HE implementation of the Transformer architecture. The main problems of this approach are the introduction of many approximations, simplified computations, and the workflow in which the client has to get involved in some computations (especially in the evaluation of some activation functions). One of our main contribution is, on the other hand, to outsource all the computational tasks to the server.

Recently, [28] introduced Primer, a privacy preserving architecture which allows fast Transformer inference. In that work, HE is used for polynomial operations and MPC for non-polynomial operations, so the whole architecture is hybrid and involves multiple actors. Our architecture, on the other hand, involves only a service provider and a client.

One key point in our work is the evaluation of the softmax function, which requires the evaluation of e^x and $1/x$. In [16], the exponential function is evaluated using the limit definition by Euler, and the inverse function is approximated using Goldschmidt algorithm. In [21] e^x is approximated using the least-squares method and, as before, Goldschmidt algorithm is applied. Our proposal takes instead advantage of the Maclaurin series for e^x and of Chebyshev polynomial for $1/x$, which provides a polynomial approximation close to the best [25].

As far as we are aware, the only intersection between HE and BERT is explored in [20], where a HE-based Logistic Regression model is trained using BERT embeddings. Nevertheless, BERT is only slightly involved in such work, since the main goal is the implementation of the Logistic Regression model. The inference from BERT, on the other hand, is performed by the client, and not by the service provider, as in our proposal.

1.2 Our contribution

In this work, the following points are explored and presented:

- An implementation of the BERT encoder using FHE primitives, executed exclusively by the server. The initial embeddings, obtained as values from a look-up table, are computed by the client. To the best of our knowledge, this is the first BERT circuit based on HE.
- A *precomputation* of Layer Normalization, in which precomputed values for mean and variance are used in order to reduce the complexity of the circuit. We demonstrate that this approach does not result in a significant reduction in the model performance.
- A performance evaluation of the encrypted circuit, fine-tuned on the Stanford Sentiment Treebank (SST-2) [22] dataset, which is used for a sentiment analysis task.

1.3 The proposed setting

We propose a setting composed of two parties: a server that offers a machine learning service based on BERT and a client (or user) which asks for a classification. The communication between the server and the client (or the user) is described by Figure 1.

The client computes a set of embeddings of a tokenized sentence using look-up tables. This set is encrypted and the ciphertexts are sent to the server. The latter evaluates a HE circuit which

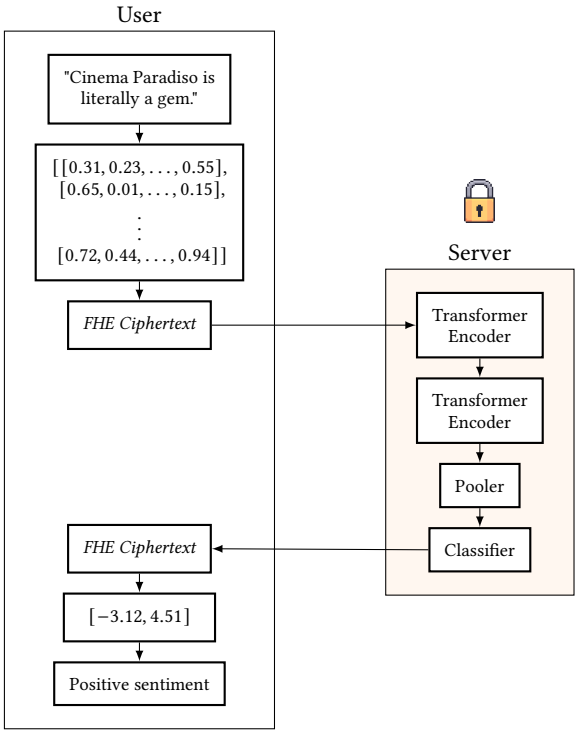


Figure 1: High-level architecture of our proposal

implements two $\text{BERT}_{\text{tiny}}$ encoders, a pooling layer and a binary classification layer. The result is thus a ciphertext containing two values, one for each output neuron. The client is able to get the result of the classification by decrypting the ciphertext and verifying whether the value contained in the first slot is greater than the value contained in the second slot.

1.4 Approximate Homomorphic Encryption

Our circuit is based on the CKKS [10] scheme, which is an approximate HE scheme that allows computations on encrypted vectors of complex numbers. Let $\mathcal{R} := \mathbb{Z}[X]/(\mathbb{Z}^N + 1)$ be a cyclotomic ring for a power of two N, and $\mathcal{R}_Q$ be the quotient ring of $\mathcal{R}$. Given a plain vector $\mathbf{v} \in \mathbb{C}^{N/2}$, the encryption informally follows the following process:

$$\mathbf{v} \in \mathbb{C}^{N/2} \xrightarrow{\text{encode}} \mathcal{R} \xrightarrow{\text{encrypt}} \mathcal{R}_Q^2$$

Since values are discretized in $\mathbb{Z}$, a parameter Δ is introduced in order to control the precision of the encoding procedure. Vectors are encoded in polynomials in such a way that polynomial multiplications result in Hadamard multiplication (i.e., slot-wise) in the clear space $\mathbb{C}^{N/2}$.

We refer to the *level* of a ciphertext as the number of multiplications that has been performed on it. A multiplication between two ciphertexts result in a ciphertext with a doubled scale equal to Δ^2; therefore, a so-called *rescaling* operation is performed. Nevertheless, this operation reduces the modulo Q of the polynomial by multipliying by a factor Δ^{-1}.

The modulus $Q = q_1 \cdot q_2 \cdot \ldots q_n$ is built as a *moduli chain* using multiple q_i, so that when a rescaling is performed, the modulo Q "loses" a modulus q_i. When Q reaches the minimum level $Q = q_1$, it can not be rescaled anymore. A *bootstrapping* operation is required in order to bring back the modulus to the original Q (equivalently, to reduce the level of a ciphertext). We use the bootstrapping technique described in [8]. We refer the reader to [19] for further information about the workings of the CKKS scheme.

This paper uses the following primitives, using OpenFHE API notation:

- EvalAdd(ct_1, ct_2): performs a slot-wise addition between two ciphertexts/plaintexts ct_1 and ct_2.
- EvalMult(ct_1, ct_2): performs a slot-wise multiplication between two ciphertexts/plaintexts ct_1 and ct_2.
- EvalRotate(ct, i): the positions of the encrypted values in ct are rotated to the left by i positions.
- EvalBootstrap(ct): performs the bootstrap operation on the ciphertext ct.

Since the bootstrapping operation theoretically enables the evaluation of circuits of any depth, the term Fully Homomorphic Encryption (FHE) is used.

2 METHODOLOGY

This section contains the design of the proposed FHE circuit, and the definition of the basic algorithms on which the circuit is based.

2.1 Some basic HE algorithms

Efficient computations in a HE circuit, particularly matrix multiplications, require adequate methods for encoding and packing data. In literature, it is possible to find many approaches to vector-matrix multiplication. For instance, [13] presented a so-called diagonal form for multiplications, while [18] an even better hybrid approach.

When building an encrypted version of a feed-forward or convolutional neural network, it is usually possible to apply any pre-processing (i.e., moving and repeating elements) to weight matrices, since they are almost always stored as plaintexts. Nevertheless, in the Self-Attention layer of Transformers, many matrix multiplications are performed between matrices that can not be pre-processed. For instance, the first multiplication is performed between **Q** and **K** which are, in turn, obtained as matrix multiplications. In this case pre-processing is not possible, and reshaping ciphertexts is an expensive operation. We therefore propose two algorithms to perform matrix-vector multiplication, depending on how the arguments are packed. The main idea is that, in order to avoid reshaping, procedures are adapted to data, and not vice versa. Given a vector of length k, and ciphertexts of s slots, where $k < s$ and both are powers of two, we define:

- Repeated packing: the plain vector is repeated k/s times along the ciphertext.
- Expanded packing: each of the k elements of the plain vector is repeated k/s times.

This is a *required redundancy*, needed to take full advantage of SIMD computations. Notice that in our implementation k is equal to the hidden size H of the model (in $\text{BERT}_{\text{tiny}}$ the hidden size is $H = 128$), while the number of slots in each ciphertext is $s = 2^{14}$. Matrices, on the other hand, are encoded in Row-major packing or in Column-major packing.

2.1.1 *Vector-matrix multiplications.*

We introduce two distinct procedures, namely VecMatER and VecMatRC. Both work on vectors of length n and square matrices of size $n \times n$, and they share the same semantic. Plus, they both rely on the RotSum procedure, that given b and t as input, sums the elements in positions $\{i \cdot b : 0 \leq i < t\}$ by rotating the input ciphertext. The first one, VecMatER, is performed between an Expanded vector and a Row-major matrix (as the name suggests), and is presented in Algorithm 1.

Algorithm 1 Vector-Matrix in Expanded and Row-major shapes multiplication

```
1: procedure VecMatER(ct_1, ct_2)
2:     ct_res ← EvalMult(ct_1, ct_2)
3:     ct_res ← RotSum(ct_res, t = 128, b = 128)
4:     return ct_res
5: end procedure
```

Figure 2 provides a visual representation of the procedure.

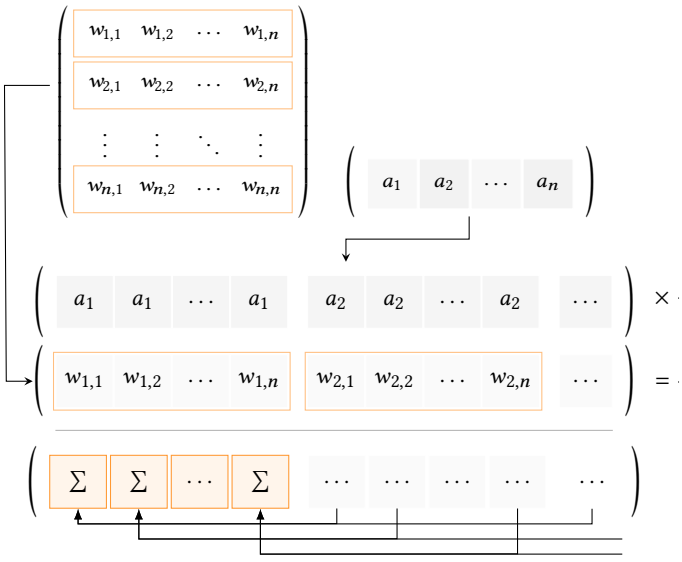


Figure 2: Visual representation of the VecMatER procedure

The procedure performs only one multiplication, meaning that only one level is consumed. The output is a ciphertext in Repeated shape, ready to be used by the following procedure, without any reshaping. The VecMatRC procedure takes as input a Repeated vector and a Column-major matrix. It is described in Algorithm 2 and it is represented in Figure 3. The output of this algorithm is a ciphertext that contains the resulting values in positions $i : i \bmod 128 = 0$. By repeating these values 128 times, we obtain an Expanded ciphertext, that can be ideally used again by the VecMatER procedure. Since the repetition of values requires a mask procedure, this procedure consumes a total of two levels.

Algorithm 2 Vector-Matrix in Repeated and Column-major shapes multiplication

```
1: procedure VecMatRC(ct_1, ct_2)
2:     ct_res ← EvalMult(ct_1, ct_2)
3:     ct_res ← RotSum(ct_res, t = 128, b = 1)
4:     ct_res ← Repeat(ct_res, t = 128, b = 1)
5:     return ct_res
6: end procedure
```

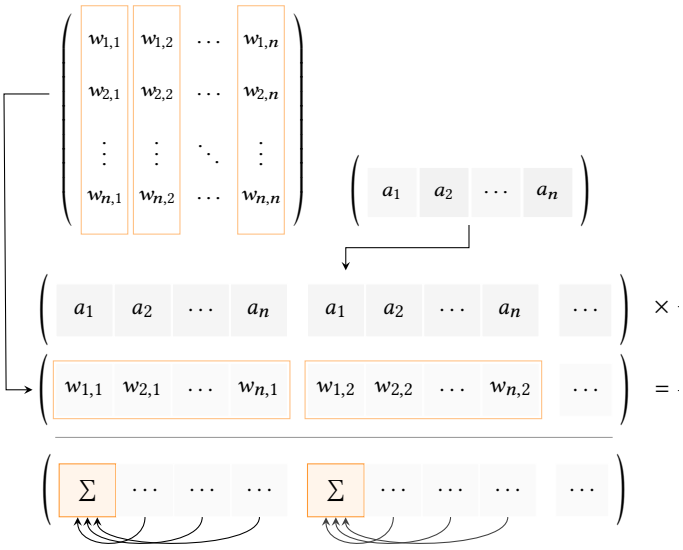


Figure 3: Visual representation of the VecMatRC procedure

2.1.2 Vector wrapping algorithms. We present two procedures that, combined with vector-matrix multiplications, enable actual matrix-matrix multiplications. The idea is to perform n times a vector-matrix multiplications and to wrap the results up in a single ciphertext. The first procedure is called WrapUpExpanded, and it is used to wrap at most n Expanded shape ciphertexts, see Algorithm 3 and Figure 4 for a visual representation. The output of this procedure

Algorithm 3 Wrap up Expanded

```
1: procedure WrapUpExpanded(v, n)
2:     for i ← 0 to len(v) − 1 do
3:         v_masked[i] ← MaskMod(v_i, i, n)
4:     end for
5:     return EvalAdd(v_masked)
6: end procedure
```

is a ciphertext containing a Column-major matrix. The MaskMod procedure takes as input an index i and selects the values in positions j such that $j \equiv i \bmod n$ by applying a binary mask encoded as a plaintext. All the masked ciphertexts are then summed using the EvalAdd procedure.

On the other hand, WrapUpRepeated is used to wrap n Repeated shape ciphertexts, where n is the number of repetitions. The output is a ciphertext containing a Row-major matrix consisting

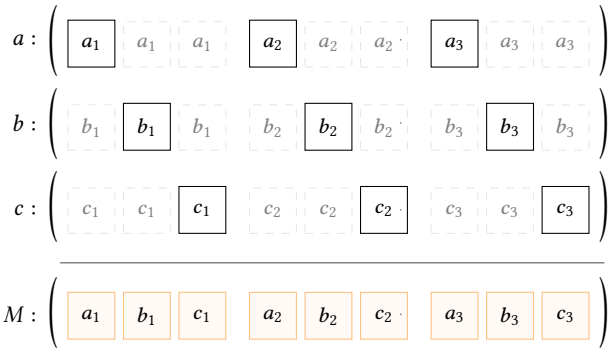


Figure 4: Visual representation of the WrapUpExpanded procedure; M is the resulting Column-major matrix

of the n row vectors; see Algorithm 4 and Figure 5 for a visual representation. The MaskBlock procedure is used to extract the

Algorithm 4 Wrap up Repeated

```
1: procedure WrapUpRepeated(v, n)
2:     for i ← 0 to len(v) − 1 do
3:         v_masked[i] ← MaskBlock(v_i, n · i, n · (i + 1))
4:     end for
5:     return EvalAdd(v_masked)
6: end procedure
```

i-th repetition of the i-th ciphertext, as shown in Figure 5. In the actual implementation the value of n is always equal to 128.

Figure 5: Visual representation of the WrapUpRepeated procedure; M is the resulting Row-major matrix

2.2 Circuit design

A classification model based on BERT is composed of four parts: Embeddings, Encoders, Pooler and Classifier. The first one (Embeddings) consists of a couple of lookup tables and a normalization, and is executed by the client, which stores the tables for $31036 \cdot 128$ float real numbers (a total of $\approx$ 16MB). The other three parts are homomorphically evaluated server-side, and this section introduces and analyzes them as a circuit based on FHE primitives.

Each Encoder is composed by four sequential parts: *BertSelfAttention*, *BertSelfOutput*, *BertIntermediate* and *BertOutput*.

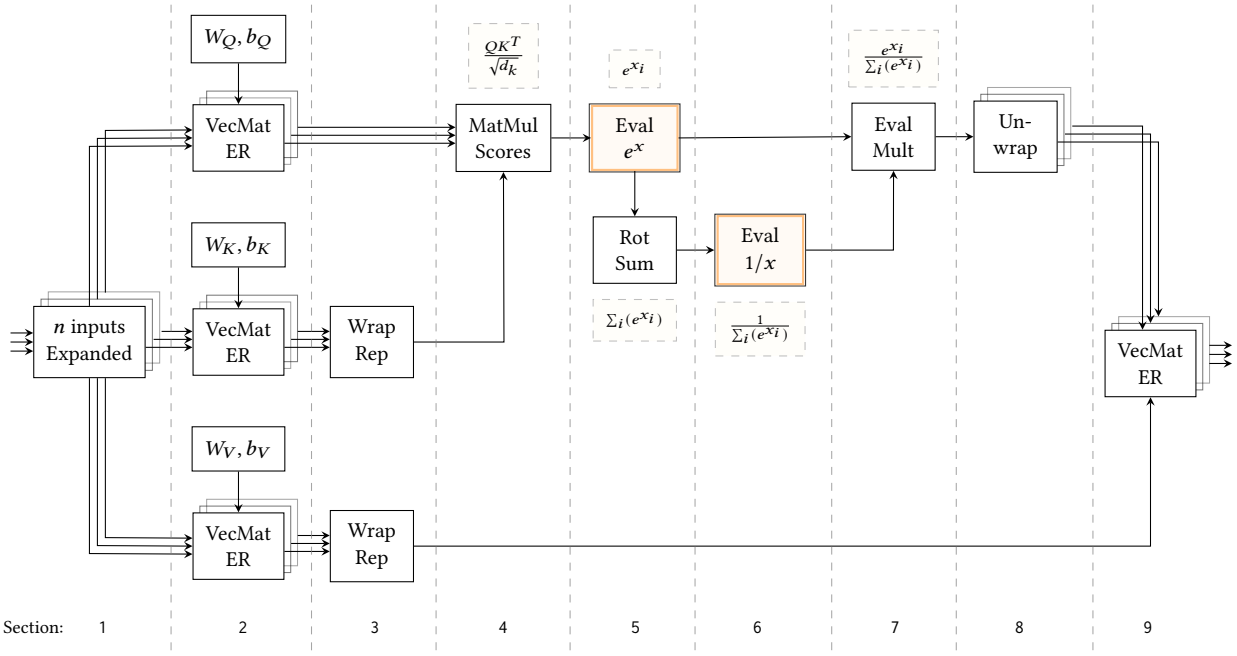


Figure 6: Circuit diagram illustrating the HE implementation of the BertSelfAttention layer

2.2.1 BertSelfAttention. The first layer computes the following function:

$$\text{Self-Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{1}$$

where $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are the query, key and value representations of the input, respectively, obtained as a result of an affine transformation $Ax + b$. The dimension d_k of the keys is, in the case of $\text{BERT}_{\text{tiny}}$, equal to 64. Lastly, $\text{softmax}(x)$ is defined as:

$$\text{softmax}(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2}$$

In particular, e^x is evaluated in two phases. First, an interval $[-r, r]$ that contains all possible input values is experimentally determined. To prevent instability when working with large numbers, we approximate e^x as $e^{x/r}$ in the interval $[-1, 1]$, then the result is raised to the power of r, obtaining $(e^{x/r})^r = e^x$. We will use n to indicate the number of input tokens. Figure 6 gives a visual representation of the HE circuit of *BertSelfAttention*, whose sections are explained below.

- *Section 1* – this is the input of the algorithm: n ciphertexts representing n token embeddings in Expanded packing shape.
- *Section 2* – the matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are derived by evaluating an affine transformation $Ax + b$ using the respective weights and biases. In particular, each VecMatER operation returns a set of n ciphertexts, each representing a row of the resulting matrix.
- *Section 3* – the ciphertexts containing the rows of the key matrix $\mathbf{K}$ are wrapped up in a single ciphertext using the WrapUpRepeated procedure. We proceed in the same way with the rows of $\mathbf{V}$. This produces two matrices encoded in Column-major order.
- *Section 4* – at this stage the so–called *scores* (in particular, the dot product attention scores) are computed. The MatMulScores procedure takes as input the matrix $\mathbf{K}$, stored in a single ciphertext in Row-major packing, and the vectors $\mathbf{Q}_i$ (with $0 \le i < n$) containing the rows[1] of matrix $\mathbf{Q}$. It computes VecMatRC($\mathbf{K}$, $\mathbf{Q}_i$) and wraps the results up. The masking phase is done using $(1/8) \cdot (1/r)$ as the mask value: the first term reproduces the $\sqrt{d_k}$ division, the second one prepares the values for the upcoming $e^{x/r}$ approximation.
- *Section 5* – the two heads, contained in a single ciphertext, are given as input to the first seven terms of the Maclaurin series of e^x. In particular, as stated above, the approximated function is $e^{x/r}$ in the interval $[-1, 1]$. The result is then raised to the power of r, consuming $\log(r)$ levels. Next, the two heads are wrapped in a single ciphertext that is then cloned: the first copy is kept as-is, while the second one is given as input to the RotSum procedure. At this point, the sum of all the required terms e^{x_i} for the k-th softmax denominator (Eq. (2)) is placed in positions i such that $i \equiv k \bmod 128$, as shown in Figure 8.

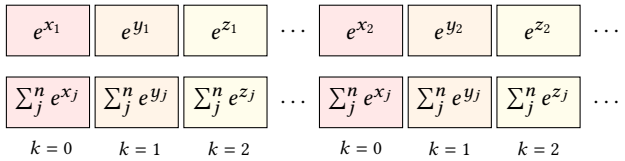


Figure 8: Example of ciphertexts content after RotSum of e^x

[1] Eq. (1) shows that the matrix $\mathbf{K}$ is transposed, but transposing is not required since the matrix is in Row-major order as a consequence of the first VecMatER procedure.

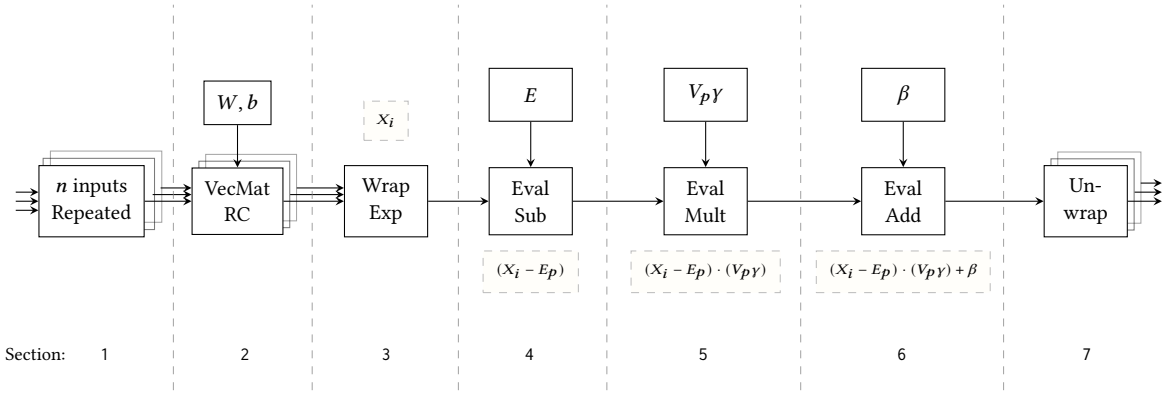


Figure 7: Circuit diagram illustrating the HE implementation of the BertSelfOutput layer

We use $0 \leq k < n$ to index the different denominators, since the ciphertext wraps all the n inputs.

- *Section 6* – since each softmax requires the inverse of the sum $\sum_{j=1}^{n} e^{x_j}$, these values are given as input to a polynomial approximation of $1/x$.
- *Section 7* – to finalize the division, the numerators e^{x_i} (the first clone) are multiplied by $1/\sum_{j=1}^{n} e^{x_j}$.
- *Section 8* – n vectors are unwrapped in separate ciphertexts. The UnwrapExpanded procedure does the opposite of WrapUpExpanded.
- *Section 9* – using the VecMatER procedure, the n Expanded vectors obtained by unwrapping softmax results are multiplied by **V**. Notice that the results will be in Repeated shape.

2.2.2 BertSelfOutput. This layer computes a dense layer followed by a Layer Normalization [5], which is defined as:

$$\text{LayerNorm}(X) = \frac{X_i - \text{E}[X]}{\sqrt{\text{Var}[X] + \varepsilon}} \cdot \gamma + \beta \tag{3}$$

Each X_i, with $0 \leq i < 128$, represents a feature of the input token X. $\text{E}[X]$ is the mean of these values, while $\text{Var}[X]$ the variance. The weight and the bias of the layer are referred to γ and β, respectively.

One of the main bottlenecks when building HE-based circuits is the evaluation of non linear functions; notice that the Layer Normalization implies the evaluation of an inverse square root. Taking inspiration from Residual Networks [14], where the Batch Normalization is evaluated using precomputed values, we implemented a *Precomputed Layer Normalization*, where the values of $\text{E}[X]$ and $1/\sqrt{\text{Var}[X] + \varepsilon}$ are experimentally observed and precomputed. We therefore computed vectors of mean values (represented as blue and red lines in Figure 9) and simplified Eq. (3) as follows:

$$\text{LayerNorm}(X) = (X_i - E_p) \cdot (V_p\gamma) + \beta \tag{4}$$

where E_p is the precomputed mean vector for $\text{E}[X]$, and V_p is the precomputed mean vector for $1/\sqrt{\text{Var}[X] + \varepsilon}$. See Figure 9 for a visual example. This approach simplifies a lot the circuit, and it does not significantly affect the accuracy of the classifications, as will be shown in the experiments presented in Section 3. Let us now describe each section of Figure 7.

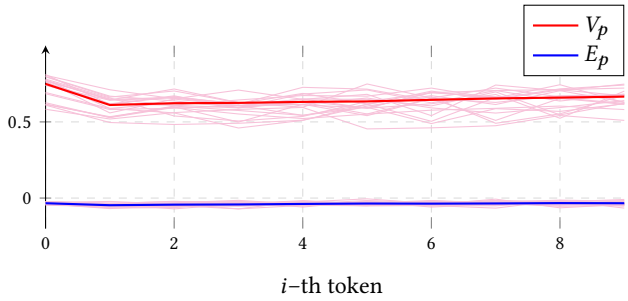


Figure 9: Example of distribution of $1/\sqrt{\text{Var}[X] + \varepsilon}$ (top) and $\text{E}[X]$ (bottom) for the first ten tokens in the very first Layer Normaliztion

- *Section 1* – n Repeated ciphertexts coming from the previous layer.
- *Section 2* – the n MatVecRC procedures evaluate the first dense layer. The output rows are in Expanded shape.
- *Section 3* – a wrapping operation is executed in order to create a single ciphertext containing all the X_i needed by the following blocks.
- *Section 4* – the computation of Eq. (4) starts here. The value of $X_i - E_p$ is obtained by using the precomputed mean values for $\text{E}[X]$. By taking advantage of SIMD computations, this is feasible in a single subtraction.
- *Section 5* – the previous result is then multiplied by $V_p\gamma$, which contains the precomputation of $\gamma/\sqrt{\text{Var}[X] + \varepsilon}$.
- *Section 6* – the evaluation of the Layer Normalization ends with the addition of the bias β.
- *Section 7* – The vectors contained in the ciphertext are unwrapped in n Expanded ciphertexts.

2.3 BertIntermediate

This layer is composed of two parts: a dense layer with 512 hidden units and the Gaussian Error Linear Unit (GELU) [15] activation function.

The first matrix multiplication needs further investigation, since the size of the weight matrix is 128×512, which of course can not

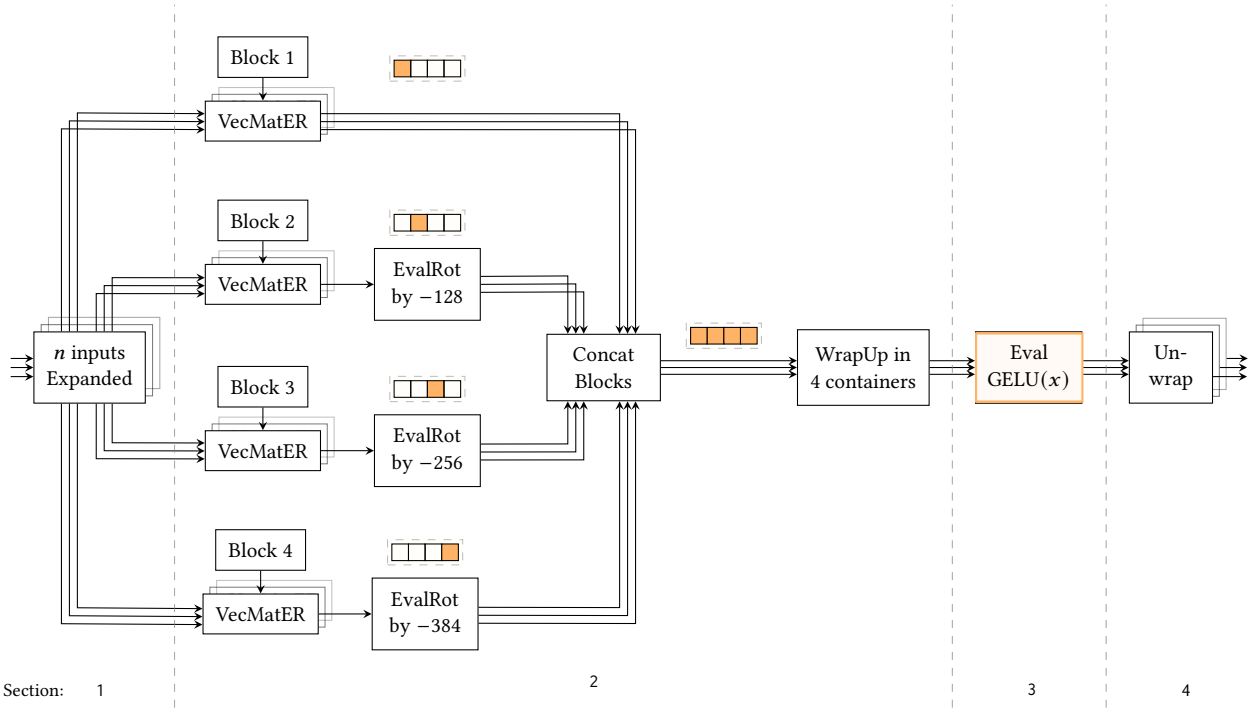


Figure 10: Circuit diagram illustrating the HE implementation of the BertIntermediate layer

be encoded in a single plaintext. The approach is to split the matrix in four parts, so that they can be individually encoded. Remark that the input values are Repeated, meaning that the vector-matrix procedure to be used is VecMatER. The weight matrix, thus, must be split by columns and encoded by rows (Figure 11).

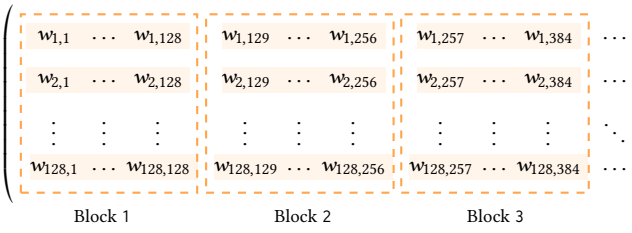


Figure 11: Splitting a 128×512 matrix into four 128×128 square matrices. The dashed squares represent the split over the columns, the highlight represents the Row-major encoding

This approach allows one to perform the actual vector-matrix multiplication by executing four times VecMatRC, each time with a different block. The first procedure returns the first 128 results, the second procedure the second 128 results, and so on.

All the obtained values will then be concatenated and fed as input to the GELU(x) activation function, which is defined as:

$$\text{GELU}(x) = x \cdot \frac{1}{2}(1 + \text{erf}(x/\sqrt{2})) \tag{5}$$

where erf(x) is the so-called Error Function erf(x), defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t \tag{6}$$

In the following, all the sections from Figure 10 will be reviewed.

- *Section 1* – n Expanded ciphertexts received as input from the previous layer.
- *Section 2* – the VecMatER procedure is executed four times, one for each block of the weight matrix. The results obtained in the 128 positions are masked, so that the values can later be concatenated. Specifically, the first part calculates values ranging from 0 to 127, the second part computes values from 128 to 255, and so on. Rotations are needed because results are always located in the first 128 slots, hence they need to be shifted to be merged. By combining these parts together, we obtain the complete set of 512 resulting values for each vector.
- *Section 3* – the $4n$ ciphertexts need to be wrapped before evaluating GELU(x). Since each vector now contains 512 values, a single ciphertext will not be enough. Thus, we build four containers containing the wrapped up values. In general, $\lfloor n/32 \rfloor$ containers are required. Since results are already masked, the wrapping operation can be performed without further masking (hence, no levels consumption). Lastly, it is possible to evaluate GELU(x) on each container.
- *Section 4* – the results are unwrapped in $4n$ ciphertexts, four for each input. In particular, each block of 128 values in each ciphertext is stored in Repeated packing shape.

2.3.1 BertOutput. This is the last layer of the Transformer Encoder. It first reduces the number of features for each input from 512 back

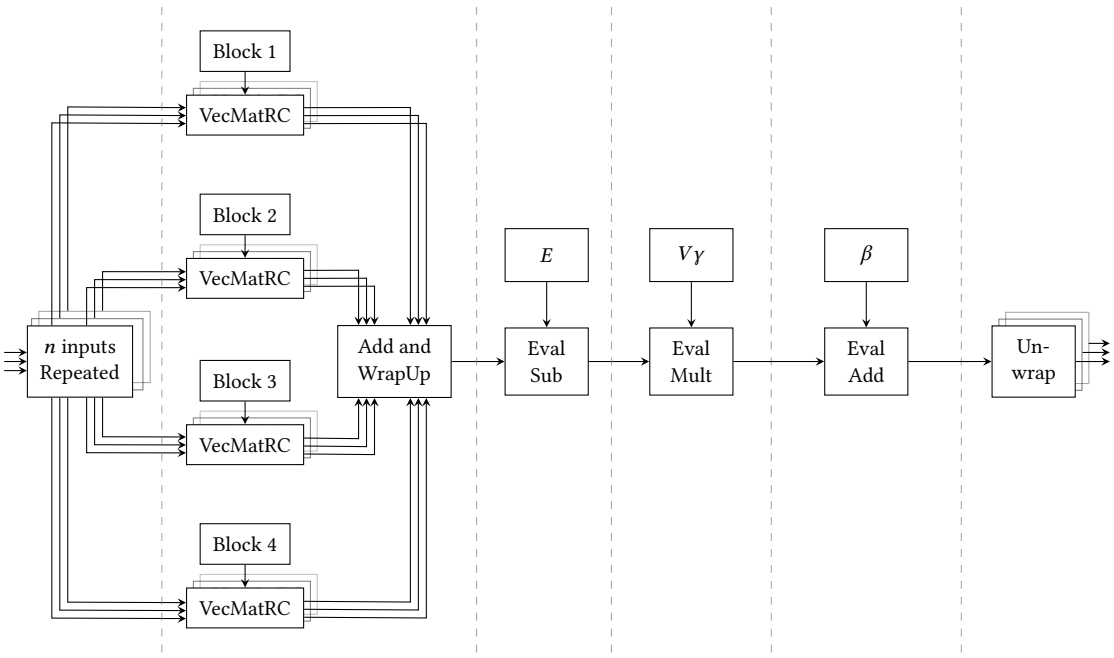


Figure 12: Circuit diagram illustrating the HE implementation of the BertOutput layer

to 128 by evaluating a dense layer using a weight matrix of size 512×128. Like in the previous layer, performing this operation is not trivial because the weight matrix does not fit a single plaintext. Since the input ciphertexts are packed in Repeated mode, the corresponding matrix-vector multiplication to be performed is VecMatRC. The weight matrix is thus split along the rows and encoded by columns. Each column is split into four parts, and the results of these four multiplications must be added together in order to get the final results. In doing so, we obtain n ciphertexts in Expanded packing shape.

The subsequent part of the layer is a precomputed Layer Normalization, which has already been analyzed in Section 2.2.2 under the same input shapes (n Expanded ciphertexts).

2.3.2 Pooler. After two stacked encoders, $\text{BERT}_{\text{tiny}}$ implements a pooling layer, whose purpose is to provide a fixed-size representation of the entire input sequence. In particular, the Pooler layer operates on the final layer's hidden states and uses the output of the special [CLS] token, which is inserted at the beginning of the input sequence by the tokenizer. A fully connected layer is then evaluated on this vector, followed by the computation of the hyperbolic tangent $\tanh(x) = \sinh(x)/\cosh(x)$ activation function. It is possible to obtain a close approximation of this function by utilizing Chebyshev polynomials, even with a relatively low degree, since the observed interval of approximation required is $[-20, 20]$, which is pretty small.

2.3.3 Classifier. The output of the Pooler layer is a ciphertext in Repeated shape. The last operation is a VecMatRC procedure, in order to evaluate the last Classifier layer, composed of a single dense layer with two output neurons. This is the output of the circuit, which is sent back to the client.

3 EXPERIMENTS

We now aim to assess the performance of the FHE circuit. All the experiments have been carried out according to the security level of 128 bits established by the Homomorphic Encryption Standards [4]. The reference dataset is the Stanford Sentiment Treebank (SST-2), which is a dataset composed of 67.3k training sentences extracted from movie reviews, each labeled with a positive or negative sentiment. The FHE circuit has been evaluated on a M1 Pro CPU, and it requires approximately 18GB of memory. The source code used for the experiments is freely available as an open-source repository[2].

3.1 Parameters

The parameters for the CKKS scheme have been chosen considering two constraints: $\lambda = 128$ bits of security and ciphertexts containing $s = 2^{14}$ values. The set of parameters is the following:

- Ring of dimension $N = 2^{16}$
- Ciphertext slots $s = N/4$
- Precision factor $\Delta = 55$ bits
- Moduli chain values $q_i = 52$ bits
- Circuit depth $d = 24$
- Ciphertext modulus $Q = 1767$ bits

We refer the reader to Kim et al. [19] for the details about the parameters. Since each embedding spans 128 dimensions, the circuit is able to handle at most $s/128 = 128$ tokens.

At first, the plain $\text{BERT}_{\text{tiny}}$ circuit has been evaluated on the whole training set, and the inputs of the non-linear functions have been experimentally observed. This step is required in order to

[2] https://github.com/narger-ef/FHE-BERT-Tiny

determine the intervals in which Chebyshev polynomials will approximate the non-linear functions, and also to determine their degree, see Table 1.

Table 1: Chebyshev polynomials parameters of approximated non-linear functions in pre-trained HE model

Layer	Function	Interval	Degree
Encoder 1 – Self-Attention	$1/x$	$[2, 5000]$	119
Encoder 1 – Intermediate	GELU(x)	$[-14, 11]$	119
Encoder 2 – Self-Attention	$1/x$	$[3, 130000]$	200
Encoder 2 – Intermediate	GELU(x)	$[-18, 8]$	59
Pooler	$\tanh(x)$	$[-20, 20]$	300

3.2 Plain circuit

The chosen plain $\text{BERT}_{\text{tiny}}$ model[3] has been fine tuned on the sentiment analysis task over the SST-2 training set, and it achieved an accuracy of 0.837 on the validation set (we remark that the test set labels are not publicly available).

The first experiment aims to understand the performance loss introduced by using precomputed values for mean and variance in Layer Normalization. As discussed in Section 2.2.2, we first obtained mean vectors for $\mathrm{E}[X]$ and $1/\sqrt{\mathrm{Var}[X] + \varepsilon}$ using the training set, then we used these values when evaluating the model on the validation set.

A performance drop from 0.837 to 0.815 has been observed, which is considered to be acceptable for our task, since the introduction of precomputed Layer Normalization allows one to avoid the evaluation of four inverse square roots (two for each encoder), resulting in faster encrypted computations.

3.3 Encrypted circuit

The encrypted circuit is evaluated server-side, and the client requests for a classification as shown in Figure 1.

In our setting, the client has to tokenize the sentence and to encrypt the initial embeddings. The ciphertexts are then sent to the service provider, which we assume to be an honest-but-curious server. Then, the FHE circuit is evaluated and the output of the network will be a ciphertext containing two values. This is decrypted by the user, that is able to see the result of the classification.

When building a deep FHE circuit using a so-called *levelled* scheme[4], one must carefully consider the ciphertexts level growth, and choose when to perform the bootstrapping operations. In particular, this procedure must be performed when all the embeddings are wrapped in a single ciphertext and ideally when values are close to the interval $[-1, 1]$, in order to obtain more accurate results.

Figure 13 presents how the level of the ciphertexts grows during the evaluation of the circuit, and where we decided to perform a bootstrapping. The operation is not always performed at the last level (24) because it could happen that, at that level:

- The embeddings are not wrapped in a single ciphertext.

[3]Available at `https://huggingface.co/philschmid/tiny-bert-sst2-distilled`
[4]A levelled scheme allows for a fixed number of multiplications (in our case d) before requiring a bootstrapping.

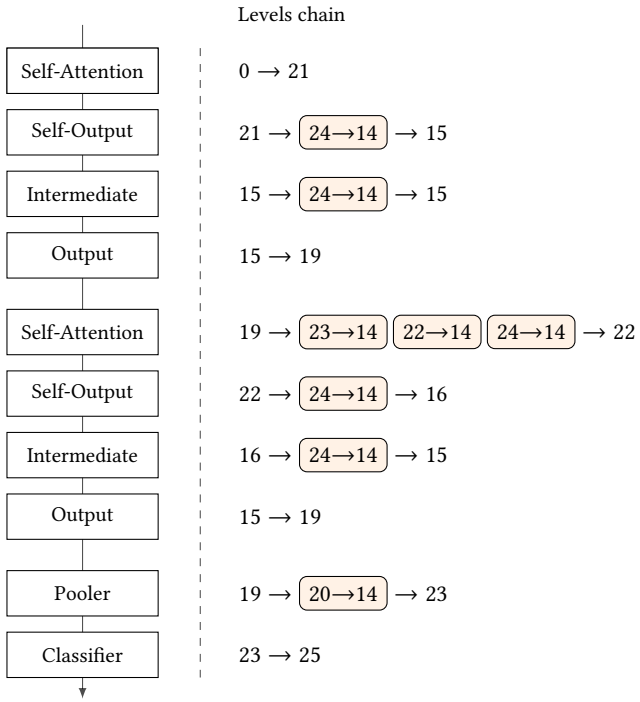


Figure 13: Ciphertexts levels progress during the evaluation of the circuit. An orange block means that a bootstrapping operation is performed.

- There is an evaluation of a Chebyshev polynomial, which can not be discontinued by a bootstrapping operation because of the recursive nature of the Paterson-Stockmeyer [23] polynomial evaluation algorithm.
- The values of the ciphertext are much larger than 1.

If one of these cases occur, bootstrapping must be performed beforehand.

To begin with, we first want to give an interpretation of the result given by the FHE circuit. The first aspect that is analyzed is indeed the error introduced by the approximate computations, and how it impacts the final classification. We remark that the classification is performed by finding the most active between the two final neurons (i.e., which one contains the largest value). Therefore, a classification is considered to be correct unless the approximations cause an inversion of the order between the two output values. A visual representation of the impact of approximations is given in Figure 14.

More formally, a classification is incorrect if $d_n < (e_1 - e_2)/2$. In that case, n_1 might become larger than n_2 (or vice versa). To summarize, the probability to obtain a correct classification depends on:

- The amount of error introduced by approximations ($(e_1 - e_2)/2$).
- The distance between the values of the two output neurons (d_n).

The first aspect can be controlled by carefully selecting the parameters Δ and q_i of the CKKS scheme (defined in Section 3.1) and by

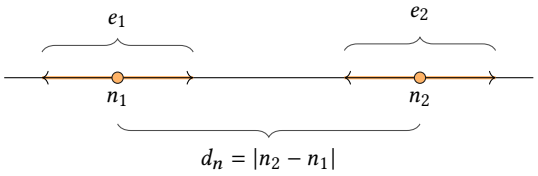


Figure 14: Example of two output values n_1, n_2 and errors e_1, e_2 added by the approximate FHE circuit.

the approximations of the non-linear functions (see Table 2). The second aspect, on the other hand, is an intrinsic characteristic of the model.

Before presenting the results, we want to emphasize that small approximations are not always indicative of correct classification, and vice versa. For example, by considering the sentence "*it 's a work by an artist so in control of both his medium and his message that he can improvise like a jazzman.*", the output values have a distance $d_n \approx 0.018$. This could happen because the features extracted by $\text{BERT}_{\text{tiny}}$ are not enough for the classifier to well distinguish between the two sentiments. In this case, even a small error could cause a wrong classification. On the other hand, the output values for the sentence "*the best film about baseball to hit theaters since field of dreams.*" have a distance $d_n \approx 3.897$, meaning that even large errors could result in correct classification. We define the function used to evaluate the overall error between the expected (plain) and obtained (FHE) values as:

$$\text{error}(v, u) = \sum_{i=1}^{|v|} \left(\left| \frac{v_i - u_i}{max(v)} \right| \right) \cdot ||v||^{-1} \tag{7}$$

where v represents the expected vector, and u the obtained one. We present in Figure 15 the distances between the output values (d_n) computed in plain on the entire validation set, and the errors obtained by the FHE circuit, evaluated on the same set.

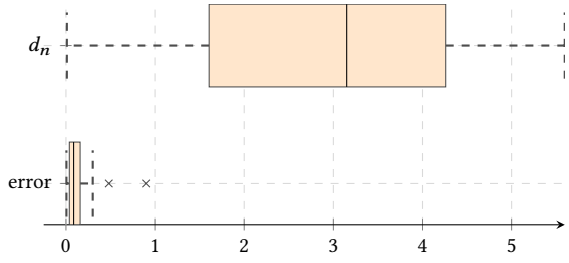


Figure 15: Errors, relative to the FHE circuit, and the distance between the output values (d_n) obtained from evaluating the validation set.

We observed that the main cause of errors is the Chebyshev approximation of the $1/x$ function, evaluated in the Self-Attention layer of the second encoder, since its approximation interval is very large, as shown in Table 1. Nevertheless, errors are usually below the output values, meaning that the FHE circuit should achieve a similar performance to the plain one. To complete the evaluation, we present the accuracy obtained by the FHE circuit in Table 2. The loss in accuracy, with respect to the $\text{BERT}_{\text{tiny}}$ model with precomputed Layer Normalization, is -0.025.

Table 2: Accuracy for the three considered models, on the SST-2 validation set. LN stands for Layer Normalization

Model	Accuracy	Loss
$\text{BERT}_{\text{tiny}}$	0.837	(*reference*)
$\text{BERT}_{\text{tiny}}$ with precomputed LN	0.815	0.022
FHE-$\text{BERT}_{\text{tiny}}$	0.790	0.047

Then, we concentrate on computational times; in particular, we want to highlight the correlation between the execution time and the number of tokens n.

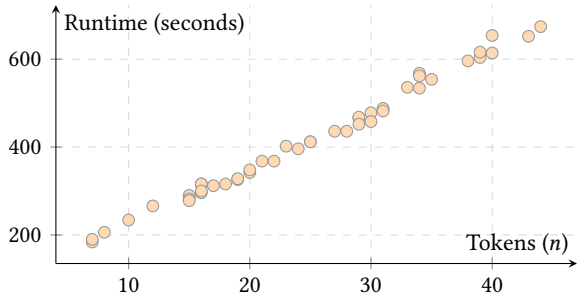


Figure 16: Linear correlation between the FHE circuit runtime and the number of tokens

As confirmed by Figure 16, there is a strong linear correlation between computational time and the number of tokens.

4 CONCLUSION

In this paper, we proposed a circuit which implements $\text{BERT}_{\text{tiny}}$ [7] based on FHE primitives, precisely on the RNS-CKKS scheme [19]. In particular, the setting is composed of an honest-but-curious server that offers the language model as a service to a client, which has to tokenize and create token embeddings of a sentence. These vectors are then encrypted and sent to the service provider, that computes the Transformer encoder layers, a pooling layer and a classification layer in a privacy-preserving environment. The circuit has been implemented using the OpenFHE library on a M1 Pro CPU, and it requires 18GB of RAM. The source code of the circuit is available at our open-source repository. Results are promising, although this proposal is a baseline from which various improvements can be carried out. First of all, it would be interesting to scale the architecture to support larger BERT models, but also to test this circuit on more accurate polynomial approximations of activation functions, to see how the results change. In HE literature it is possible to find references to HE-friendly networks, in the sense that they are not based on non-linear functions. Considering a wider context, it would be also interesting to study a class of neural networks which output well-separated logits, so that the same results could be obtained using less accurate HE computations. This would translate in faster and lighter calculations.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.* 51, 4, Article 79 (jul 2018), 35 pages.

[2] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (Los Angeles, CA, USA) *(WAHC'22)*. Association for Computing Machinery, New York, NY, USA, 53–63. https://doi.org/10.1145/3560827.3563379

[3] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. 2022. Status report on the third round of the NIST post-quantum cryptography standardization process. *US Department of Commerce, NIST* (2022).

[4] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard.* Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML]

[6] 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.

[7] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. 2021. Generalization in NLI: Ways (Not) To Go Beyond Simple Heuristics. arXiv:2110.01518 [cs.CL]

[8] Jean-Philippe Boussat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Habaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Advances in Cryptology – EUROCRYPT 2021*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer International Publishing, Cham, 587–617.

[9] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022. THE-X: Privacy-Preserving Transformer Inference with Homomorphic Encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland, 3510–3520. https://aclanthology.org/2022.findings-acl.277

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer International Publishing, Cham, 409–437.

[11] Pierre-Emmanuel Clet, Oana Stan, and Martin Zuber. 2021. BFV, CKKS, TFHE: Which One is the Best for a Secure Neural Network Evaluation in the Cloud?. In *Applied Cryptography and Network Security Workshops*, Jianying et al. Zhou (Ed.). Springer International Publishing, Cham, 279–300.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv* abs/1810.04805 (2019).

[13] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *Advances in Cryptology – CRYPTO 2014*, Juan A. Garay and Rosario Gennaro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 554–571.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. [n. d.]. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[15] Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). arXiv:1606.08415 [cs.LG]

[16] Seungwan Hong, Jai Hyun Park, Wonhee Cho, Hyeongmin Choe, and Jung Hee Cheon. 2022. Secure tumor classification by shallow neural network using homomorphic encryption. *BMC Genomics* 23, 1 (09 Apr 2022), 284.

[17] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain, 427–431. https://aclanthology.org/E17-2068

[18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, USA, 1651–1668.

[19] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. 2022. Approximate Homomorphic Encryption with Reduced Approximation Error. In *Topics in Cryptology – CT-RSA 2022: Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1–2, 2022, Proceedings* (San Francisco, CA, USA). Springer-Verlag, Berlin, Heidelberg, 120–144.

[20] Garam Lee, Minsoo Kim, Jai Hyun Park, Seung-won Hwang, and Jung Hee Cheon. 2022. Privacy-Preserving Text Classification on BERT Embeddings with Homomorphic Encryption. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, USA, 3169–3175.

[21] Joon-Woo et al. Lee. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054.

[22] Bo Pang and Lillian Lee. 2005. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Kevin Knight, Hwee Tou Ng, and Kemal Oflazer (Eds.). Association for Computational Linguistics, Ann Arbor, Michigan, 115–124. https://doi.org/10.3115/1219840.1219855

[23] Michael S Paterson and Larry J Stockmeyer. 1973. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.* 2, 1 (1973), 60–66.

[24] Oded Regev. 2009. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM* 56, 6, Article 34 (sep 2009), 40 pages.

[25] Lloyd N. Trefethen. 2019. *Approximation Theory and Approximation Practice, Extended Edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA.

[26] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-Read Students Learn Better: The Impact of Student Initialization on Knowledge Distillation. *CoRR* abs/1908.08962 (2019). arXiv:1908.08962

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Red Hook, NY, USA, 6000–6010.

[28] Mengxin Zheng, Qian Lou, and Lei Jiang. 2023. Primer: Fast Private Transformer Inference on Encrypted Data. arXiv:2303.13679 [cs.CR]