Secure Transformer-Based Neural Network Inference for Protein Sequence Classification

Jingwei Chen, Linhan Yang, Chen Yang, Shuai Wang, Rui Li, Weijie Miao
Wenyuan Wu, Li Yang, Kang Wu and Lizhong Dai

***Abstract*— Protein sequence classification is fundamental in healthcare and precision medicine, such as predicting protein structures and discovering novel functions. Leveraging large language models (LLMs) is greatly promising to enhance our ability to tackle protein sequence classification problems; however, the accompanying privacy issues are becoming increasingly prominent. In this paper, we present a privacy-preserving, non-interactive, efficient, and accurate protocol called encrypted DASHformer to evaluate a transformer-based neural network for protein sequence classification named DASHformer, provided by the iDASH 2024-Track 1 competition. The presented protocol is based on our solution for this competition, which won the first place. It is arguably the first secure transformer inference protocol capable of performing batch classification for multiple protein sequences in a single execution only using leveled homomorphic encryption (i.e., without bootstrapping). To achieve this, we propose a series of new techniques and algorithmic improvements, including data-driven non-polynomial function fitting, tensor packing, and double baby-step-giant-step for computing the product of multiple encrypted matrices. These techniques and improvements enable the protocol to classify 163 encrypted protein sequences in about 165 seconds with 128-bit security, achieving an amortized time of about one second per sequence.**

***Index Terms*— Homomorphic Encryption, LLMs, Privacy-Preserving Computing, Protein Classification, Transformer.**

I. Introduction

CLASSIFYING protein sequences refer to the process of organizing and categorizing proteins based on their structural, functional, or evolutionary characteristics. Proteins are made up of chains of amino acids, and the specific sequence of these amino acids determines the structure and function of the protein. Classification is fundamental in healthcare and precision medicine, such as predicting proteins' complex structures (The Nobel Prize in Chemistry 2024), predicting the function of a newly discovered protein, etc. Roughly speaking, protein sequence classification methods include those based on sequence similarity or alignment (e.g, BLAST [1]), those based on machine learning (e.g., k-mer [2]). We refer the readers to a recent survey [3] and references therein.

Since Vaswani et al. proposed the transformer architecture [4], large language models (LLMs) such as GPT [5], BERT [6] and LLaMA [7] have developed rapidly and have already demonstrated their power in numerous applications, including protein sequence classification, e.g., ProteinBERT [8] and UniDL4BioPep [9].

However, with the widespread use of LLMs, privacy concerns have inevitably become increasingly prominent. In response, some countries and corporations have restricted the use of LLMs, including, e.g., Italy [10], Amazon, Apple, etc. [11]. Given the sensitivity of biological data, how to use LLMs to classify protein sequences while protecting protein sequences' privacy is a crucial issue.

Privacy-preserving machine learning (PPML) can be used to address this problem. Currently, the technologies used in PPML mainly include secure multi-party computation (MPC) [12], differential privacy [13], homomorphic encryption (HE) [14], and trusted execution environment [15]. Among these, only MPC and HE offer cryptographically provable security. MPC requires extensive interactions among participating parties, making it less suitable for complex applications like LLMs. In contrast, HE allows logical or arithmetic operations on ciphertexts without communication and has been widely used in PPML. Nevertheless, it is well known that the computational efficiency of homomorphic encryption is not so well, especially for non-polynomial functions and for bootstrapping (an operation on ciphertext that reduces the noise near a fresh ciphertext). Therefore, whether it can practically handle privacy-preserving LLMs' inference on encrypted protein sequences is an intriguing problem.

A. iDASH 2024-Track 1

To tackle this issue, iDASH 2024 [16] launched a track of competition (hereafter referred to as iDASH 2024-Track 1), in which participants are required to design and implement a secure and efficient protocol based on HE to perform encrypted inference of a transformer-based and pre-trained LLM named *DASHformer*. The protocol works within the

Jingwei Chen, Chen Yang, Rui Li, Weijie Miao and Wenyuan Wu (corresponding author) are with Chongqing Key Laboratory of Secure Computing for Biology, Chongqing Institute of Green and Intelligent Technology, CAS and Chongqing College, University of Chinese Academy of Sciences (e-mail:{chenjingwei, wuwenyuan}@cigit.ac.cn).

Linhan Yang and Shuai Wang are with School of Information Science and Engineering, Chongqing Jiaotong University. This work was done when LHY and SW were visiting Chongqing Institute of Green and Intelligent Technology, CAS.

Li Yang (corresponding author), Kang Wu and Lizhong Dai are with Sansure Biotech, Inc. (e-mail: ylyhp@126.com).

This work was supported partly by National Key Research and Development Project of China (2020YFA0712303), Natural Science Foundation of Chongqing (2022yszx-jcx0011cstb, cstb2023yszx-jcx0008, cstb2023nscq-msx0441) and the Light of West China Program of CAS.

client-server mode, as in Fig. 1. The query data are multiple protein sequences to be classified (100 sequences in iDASH 2024-Track 1), and the ciphertexts of the query data under an HE scheme are the input from the client for the protocol. Inputs from the server are the parameters of DASHformer. At the end of this protocol, the server sends the encrypted results to the client and the client decrypts to obtain query protein sequences' classification inferred from DASHformer.

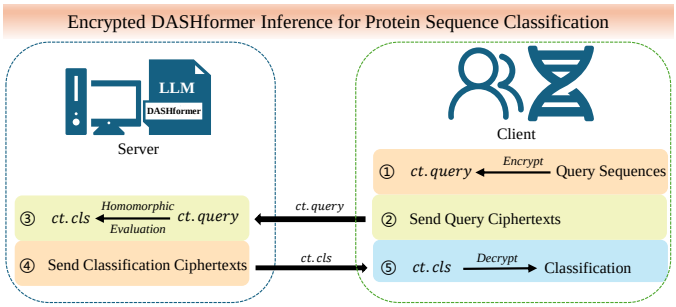


Fig. 1: The framework of our Encrypted DASHformer protocol for protein sequence classification.

DASHformer is an encoder-only transformer for protein sequence classification (see Fig. 2), consisting of layers of embedding, multi-head attention, layer normalization, feed-forward neural network (FF) and average pooling. See [17] for the details on how DASHformer is trained.

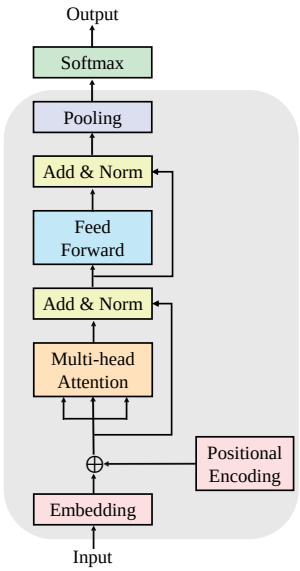


Fig. 2: Workflow of secure DASHformer inference. Modules within the gray box are required to be computed with HE.

To evaluate DASHformer on multiple encrypted protein sequences, there exist several challenges. First, DASHformer contains many non-polynomial functions, such as softmax in the attention layer, ReLU in the FF layer, and layer normalization (LayerNorm). How can we efficiently compute these HE-unfriendly functions? Second, since multiple protein sequences need to be classified, after passing through the embedding layer, the client's input is essentially a three-dimensional tensor. How can we encode this tensor so that multiple sequences can be computed in parallel rather than one by one? Third, DASHformer involves numerous plaintext-ciphertext matrix multiplications (PCMM) and ciphertext-ciphertext matrix multiplications (CCMM). How can we efficiently compute these PCMMs and CCMMs? Lastly, DASHformer has a long computational chain, which results in high multiplicative depth for encrypted computation. How can we optimize the required multiplicative depth to minimize or completely avoid the use of the expensive bootstrapping operation?

B. Our Solution

In this paper, we address the above all challenges and present a solution for iDASH 2024-Track 1, which won the first place in this competition.

More specifically, we propose a data-driven method for approximating non-polynomial functions (e.g., $\exp(x)$, $1/x$, $1/\sqrt{x}$, ReLU) in Section III. The resulting approximate functions maintain computational accuracy, and are compatible with the data packing method we adopt.

Given that the input data forms a three-dimensional tensor (each sequence is converted into a matrix after embedding and there are multiple sequences to be classified), we propose a tensor packing method named *letter-by-letter* in Section IV. Being different from the conventional row/column packing for a matrix, we pack a slice of the tensor into one ciphertext. Each slice contains one embedded vector of a letter but for all sequences. This new packing method allows us to evaluate DASHformer for multiple protein sequences in parallel. The main technical ingredients include:

- The new packing method enables us to employ an efficient PCMM algorithm that does *not* require the costly ciphertext rotation and that performs better than the one based on a naïve sequence-by-sequence packing method.
- It also directly leads to a novel technique called *double baby-step-giant-step* (*DBSGS*), which makes CCMM very efficient for three encrypted matrices. For DASHformer, this technique reduces the number of ciphertext rotations in computation of the attention layer by more than 70%, and hence reduces the running time by more than a half.

Built on these techniques, along with some further optimizations in Section V, we obtain *encrypted DASHformer*, a secure, efficient, accurate, and non-interactive DASHformer inference protocol for encrypted protein sequences without the need for bootstrapping. We implement the protocol using the Cheon-Kim-Kim-Song (CKKS) scheme [18] in `Lattigo` [19], which, under a security parameter of 128, completes the classification of 100 (but up to 163) protein sequences of length 50 within 165 seconds on a PC, achieving a micro-AUC about 0.941 for the evaluation data of iDASH 2024-Track 1; see Section VI for more details.

Although we focus on the DASHformer inference on encrypted protein sequences, we note that the presented techniques and algorithmic improvements are also beneficial to evaluate other LLMs on data encrypted by other SIMD-supported HE schemes, e.g., BGV [20] and B/FV [21], [22].

II. PRELIMINARIES

In this section, we introduce the detailed workflow of DASHformer, provide a description of the used dataset [17], and give a necessary introduction to the CKKS scheme [18].

A. Dataset

During the challenge phase of iDASH 2024-Track 1, the given dataset contains $1,000$ protein sequences along with their corresponding labels. We call it the *challenge dataset*. Each protein sequence in the challenge dataset consists of 50 one-letter amino acid symbols and belongs to one of the 25 protein families. Each amino acid symbol comes from an alphabet of size 25. For example,

$$LDLAGDPTFADLVTRTRTTLLDAQDH$$
$$EDAPFEEVVRRVAPERDPGRTPVF, 24$$

is one of the given sequence classified into the protein family with a label 24. The challenge dataset will be used to approximate the non-polynomial functions in Section III. In the evaluation phase, another test set of 100 class-balanced protein sequences was used to evaluate the presented protocol for secure inference on encrypted data in Section VI. In addition, both the challenge dataset and the evaluation dataset, together with the parameters for DASHformer were given by the organizers, and now are available in [17].

B. DASHformer

DASHformer is an encoder-only pre-trained neural network for protein sequence classification. After one-hot encoding, each input protein sequence can be converted into a matrix $\boldsymbol{X} \in \{0,1\}^{m\times l}$, which forms an input of DASHformer. Here m is the *length* of tokens and l is the size of the alphabet. Given such an $\boldsymbol{X}$ as input, DASHformer works as follows:

- Embedding and positional encoding: $\boldsymbol{X} = \boldsymbol{X}\boldsymbol{W}_e + \boldsymbol{P}$, where $\boldsymbol{W}_e \in \mathbb{R}^{l\times d}$ and $\boldsymbol{P} \in \mathbb{R}^{m\times d}$, where d is the *hidden size* (or *model dimension*). Then set $\boldsymbol{Y} = \boldsymbol{X} \in \mathbb{R}^{m\times d}$.
- H-heads attention ($0 \le h < H$, all vectors are in row):
  - Queries: $\boldsymbol{Q}^{(h)} = \boldsymbol{X}\boldsymbol{W}_Q^{(h)} + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{b}_Q^{(h)} \in \mathbb{R}^{m\times d'}$,
  - Keys: $\boldsymbol{K}^{(h)} = \boldsymbol{X}\boldsymbol{W}_K^{(h)} + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{b}_K^{(h)} \in \mathbb{R}^{m\times d'}$,
  - Values: $\boldsymbol{V}^{(h)} = \boldsymbol{X}\boldsymbol{W}_V^{(h)} + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{b}_V^{(h)} \in \mathbb{R}^{m\times d'}$,
  - Attention: $\boldsymbol{X}^{(h)} = \mathsf{softmax}\left(\frac{\boldsymbol{Q}^{(h)}\boldsymbol{K}^{(h)\mathrm{T}}}{\sqrt{d'}}\right)\boldsymbol{V}^{(h)}$,

  where $\boldsymbol{W}_Q^{(h)}$, $\boldsymbol{W}_K^{(h)}$, $\boldsymbol{W}_V^{(h)} \in \mathbb{R}^{d\times d'}$ and $\boldsymbol{b}_Q^{(h)}$, $\boldsymbol{b}_K^{(h)}$, $\boldsymbol{b}_V^{(h)} \in \mathbb{R}^{d'}$. Note that here $d' = d/H$.
- Concatenation: $\boldsymbol{X} = (\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(H-1)}) \in \mathbb{R}^{m\times d}$.
- Linear layer: $\boldsymbol{X} = \boldsymbol{X}\boldsymbol{W}_c + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{b}_c \in \mathbb{R}^{m\times d}$.
- Layer normalization: $\boldsymbol{Y} = \mathsf{LayerNorm}_1(\boldsymbol{X} + \boldsymbol{Y})$.
- FF layer: $\boldsymbol{X} = \mathsf{ReLU}(\boldsymbol{Y}\boldsymbol{W}_1 + \mathbf{1}^{\mathrm{T}}\boldsymbol{b}_1)\boldsymbol{W}_2 + \mathbf{1}^{\mathrm{T}}\boldsymbol{b}_2$, where $\boldsymbol{W}_1 \in \mathbb{R}^{d\times d''}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d''\times d}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d''}$, and $\boldsymbol{b}_2 \in \mathbb{R}^{d}$.
- Layer normalization: $\boldsymbol{X} = \mathsf{LayerNorm}_2(\boldsymbol{X} + \boldsymbol{Y})$.
- Average pooling: $\boldsymbol{x} = \mathbf{1} \cdot (\boldsymbol{X}\boldsymbol{W}_d + \mathbf{1}^{\mathrm{T}}\boldsymbol{b}_d) \in \mathbb{R}^c$, where $\boldsymbol{W}_d \in \mathbb{R}^{d\times c}$ and $\boldsymbol{b}_d \in \mathbb{R}^c$ with c the number of classes.
- Output: $\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{x})$, indicating the score of input protein sequence for each protein class.

From the above description, DASHformer is a simplified version of BERT with only one transformer layer (For instance, $\mathrm{BERT}_{\mathrm{base}}$ [6] has 12 transformer layers). The other hyperparameters of DASHformer are $m = 50$, $l = 25$, $d = 128$, $H = 4$, $d' = 32$, $d'' = 256$ and $c = 25$.

C. CKKS

The Cheon-Kim-Kim-Song scheme (CKKS) [18] is one of the most popular HE schemes. It supports approximate arithmetic operations. In CKKS, the plaintext space is a subset of $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ while messages are complex vectors in $\mathbb{C}^{N/2}$, where N is a power-of-two integer. The ciphertext space of CKKS is R/qR with a large integer q, called *ciphertext modulus*. The restriction of the canonical embedding $\mathbb{R}[X]/\langle X^N + 1\rangle \to \mathbb{C}^{N/2}$ on R maps $m(X) \in R$ into $\boldsymbol{m} \in \mathbb{C}^{N/2}$ by evaluating $m(X)$ at the primitive $2N$-roots of unity $\xi_j = \xi^{5^j}$ for $0 \le j < N/2$. The inverse of the canonical embedding encodes a message $\boldsymbol{m}$ as a plaintext $m(X)$. Thus, CKKS naturally supports single-instruction-multiple-data (SIMD) operations, i.e., performing an operation on a ciphertext corresponds to performing the same operation on $N/2$ *slots* of $\boldsymbol{m}$ in parallel.

For $\boldsymbol{x} = (x_i)_{0\le i<N/2}$ and $\boldsymbol{y} = (y_i)_{0\le i<N/2}$, let $\mathsf{ct}.\boldsymbol{x}$ and $\mathsf{ct}.\boldsymbol{y}$ be the ciphertext encrypted by CKKS under a same public key. CKKS supports the following basic operations:

- $\mathsf{Setup}(1^\lambda)$. Given a security parameter λ, output parms.
- $\mathsf{KeyGen}(\mathsf{parms})$. Output a secret key sk and the corresponding public key pk. (For convenience, we also let pk include keys for key-switching.)
- $\mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x})$. Given a message $\boldsymbol{x}$, output a ciphertext $\mathsf{ct}.\boldsymbol{x}$.
- $\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c})$. Given a ciphertext $\boldsymbol{c}$ that encrypts $\boldsymbol{x}$, output a message $\boldsymbol{x}' \approx \boldsymbol{x}$. (For simplicity, we omit sk and pk.)
- $\mathsf{Add}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})$: $\mathsf{Dec}(\mathsf{Add}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})) = \boldsymbol{x} + \boldsymbol{y}$.
- $\mathsf{Mul}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})$: $\mathsf{Dec}(\mathsf{Mul}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})) = \boldsymbol{x} \odot \boldsymbol{y}$, where $\odot$ is for component-wise multiplication.
- $\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})$: $\mathsf{Dec}(\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})) = \boldsymbol{m} \odot \boldsymbol{x}$, where $\boldsymbol{m}$ is a message in $\mathbb{C}^\ell$; for $m \in \mathbb{C}$, $\mathsf{CMul}(m, \mathsf{ct}.\boldsymbol{x})$ is a special case of $\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})$ with $\boldsymbol{m} = (m, \ldots, m)$.
- $\mathsf{Mask}_{[i,j]}(\mathsf{ct}.\boldsymbol{x})$ convert a ciphertext $\mathsf{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \ldots, x_{N/2-1})$ into a ciphertext that encrypts $(\mathbf{0}, x_i, x_{i+1}, \ldots, x_j, \mathbf{0})$, equivalent to $\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})$ for $\boldsymbol{m} = (\mathbf{0} \in \mathbb{Z}^{i-1}, \mathbf{1} \in \mathbb{Z}^{j-i+1}, \mathbf{0})$.
- $\mathsf{Rotate}_k(\mathsf{ct}.\boldsymbol{x})$ convert $\mathsf{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \ldots, x_{N/2-1})$ into a new ciphertext $\mathsf{Enc}(x_k, \ldots, x_{N/2-1}, x_0, \ldots, x_{k-1})$. Rotate costs no multiplicative depths.
  - $\mathsf{Rot}_k(\boldsymbol{x})$: If the dimension of $\boldsymbol{x}$ is strictly less than $N/2$, then rotating $\boldsymbol{x}$ with step size k, denoted by $\mathsf{Rot}_k(\boldsymbol{x})$, can be implemented with two Rotates, two Masks, and an Add. It also costs one CMul depth.

CKKS is IND-CPA secure under the RLWE assumption [23].

It is well known that the multiplicative (taking both CMul and Mul into account) depth of the task has a very significant impact on the efficiency. Additionally, as shown in [24], [25], the cost for a Rotate is several times larger than Mul. So, how to minimize the required multiplicative depth and reduce the required Rotates for a concrete application are usually the central problems in practice.

D. Adversary Model

As pointed out in Fig. 1, we consider the scenario involving two parties, namely, a client and a server. The client generates the public and private keys $(\mathsf{pk}, \mathsf{sk})$ for CKKS, while the server undertakes the computational tasks. We consider the security of our protocol under the *semi-honest* adversary model, also known as the *honest-but-curious* model [26].

III. Evaluate Non-polynomial Functions

In this section, we address the first challenge presented in the introduction, i.e., the encrypted evaluation of non-polynomial functions. The basic idea is to perform polynomial fitting of these non-polynomial functions using the challenge dataset, i.e., the $1,000$ given protein sequences. We refer to the resulting neural network *approximate DASHformer* to distinguish it from the original DASHformer.

A. ReLU

For $x \in \mathbb{R}$, $\mathsf{ReLU}(x) = \max(0, x)$. This is a piecewise function that requires comparing x with 0, which is very unfriendly to CKKS. We use a polynomial $\sum_{i=0}^{6} a_i x^i$ to approximate ReLU, where the coefficients a_i's are fitted by the least square method with the given protein sequences.

B. Layer Normalization

Given a vector $\boldsymbol{x} = (x_i)_i$,

$$\mathsf{LayerNorm}(\boldsymbol{x}) = \left(\gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma^2}} + \beta\right)_i$$

with μ and σ^2 the mean and variance of $\boldsymbol{x}$, and γ and β are parameters. Instead of computing $\frac{1}{\sqrt{\sigma^2}}$ for a ciphertext of σ^2, we compute it with a simplified form

$$\mathsf{LN}(\boldsymbol{x}) = (\gamma \cdot \tilde{\sigma} \cdot (x_i - \mu) + \beta)_i\,,$$

where $\tilde{\sigma}$ is the learned standard deviation from the given protein sequences.

In DASHformer, LayerNorm applies to all the rows of the matrix $\boldsymbol{X} \in \mathbb{R}^{m \times d}$. If we use the matrix form, we have

$$\begin{aligned}\mathsf{LN}(\boldsymbol{X}) &= \mathrm{diag}(\frac{1}{\sqrt{\sigma_i^2}})(\boldsymbol{X} - \boldsymbol{\mu}^{\mathrm{T}} \cdot \mathbf{1})\mathrm{diag}(\gamma_i) + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{\beta} \\ &=: \boldsymbol{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \boldsymbol{\Gamma} + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{\beta},\end{aligned} \tag{1}$$

where $\mathrm{diag}(\boldsymbol{a})$ is the diagonal matrix generated by the vector $\boldsymbol{a}$. Here $\boldsymbol{\mu} = (\mu_i)_{0 \le i < m}$ consists of the means of all rows of $\boldsymbol{X}$, d is the number of columns of $\boldsymbol{X}$, $\boldsymbol{\Sigma}^{-1} = \mathrm{diag}(\frac{1}{d\sqrt{\sigma_i^2}})$ and $\boldsymbol{\Gamma} = (d\boldsymbol{I} - \boldsymbol{E}) \cdot \mathrm{diag}(\gamma_i)$, where $\boldsymbol{I}$ is the identity matrix and all entries of $\boldsymbol{E}$ are 1. The parameters $(\gamma_i)_i$ and $\boldsymbol{\beta}$ are given as part of DASHformer, while $\boldsymbol{\Sigma}^{-1}$ can be fitted with the given protein sequences.

The two instances of LayerNorm that appear in DASHformer are processed using the same method described above.

C. Softmax

Softmax [27] of a vector $\boldsymbol{x} = (x_i)_i$ is defined as

$$\mathsf{softmax}(\boldsymbol{x}) = \left(\frac{\exp(x_i)}{\sum_i \exp(x_i)}\right)_i.$$

Instead of high-degree polynomial approximations of the exponential function in softmax, we use the square softmax (sqmax) [28] in the form $\frac{(x_i+c)^2}{\sum_i (x_i+c)^2}$. Furthermore, we use a constant δ to replace the denominator to avoid divisions by a ciphertext, i.e.,

$$\mathsf{sqmax}(\boldsymbol{x}) = \left(\frac{(x_i + c)^2}{\delta}\right)_i. \tag{2}$$

For the h-th head, we choose different $(c^{(h)}, \delta^{(h)})$ obtained from least square with the given protein sequences.

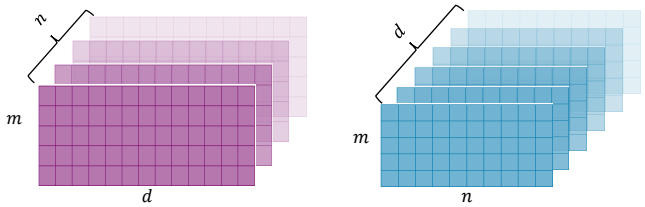
Using the sqmax function in Eq. (2) avoids the encrypted computation of the non-polynomial functions $\exp(x)$ and $1/x$. Additionally, note that sqmax is computed component-wise, which makes it compatible with any matrix encoding method, which is different from softmax. Based on this observation, we can convert the computation of $\boldsymbol{X}^{(h)}$ in the attention layer into the computation of the product of three matrices. In the next section, we will see that this makes a significant acceleration for encrypted computation of DASHformer's attention layer.

IV. Encrypted Matrix Multiplication

In this section, we focus on the computation of H-heads attention. Therefore, for convenience, we begin with the matrix $\boldsymbol{X} \in \mathbb{R}^{m \times d}$ after embedding and positional encoding.

A. Packing for Three-Dimensional Tensors

As mentioned in Section II, during the testing phase, we will classify $n = 100$ unpublished protein sequences. Testing them *sequence-by-sequence* would certainly be inefficient (see Fig. 3a). To improve efficiency, our goal is to test these n protein sequences in parallel so that we only need to run DASHformer once to complete the classification of all protein sequences. If so, we will take a $n \times m \times d$ three-dimensional tensor as the input. Instead, we adopt a packing method named *letter-by-letter packing* for three-dimensional tensors (see Fig. 3b).



(a) Sequence-by-sequence packing (b) Letter-by-letter packing

Fig. 3: Different packing methods for 3D tensor

Under letter-by-letter packing, an $n \times m \times d$ tensor will be encrypted into d CKKS ciphertexts ct_i, where ct_i encrypts the i-th $m \times n$ slice of the tensor. This slice contains the embedding vectors of the i-th letter for all n protein sequences. From a matrix perspective, ct_i actually encrypts the i-th column of all

n input $m \times d$ matrices. Keeping this in mind, we present how to perform plaintext-ciphertext matrix multiplication (PCMM) and ciphertext-ciphertext matrix multiplication (CCMM) under the letter-by-letter packing method in the next two subsections.

B. PCMM

Given ciphertexts of matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ encrypted column-wise and a plaintext matrix $\boldsymbol{W} \in \mathbb{R}^{d\times d'}$, our goal is to obtain the column-wise encrypted ciphertexts of the matrix product $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}$. Let $\boldsymbol{x}_i^{\mathrm{T}}$ be the i-th column of $\boldsymbol{X}$ and $\boldsymbol{v}_i^{\mathrm{T}}$ the i-th column of $\boldsymbol{V}$. Then $\boldsymbol{v}_j = \sum_{i=0}^{d-1} w_{i,j}\boldsymbol{x}_i^{\mathrm{T}}$ for $0 \le j < d'$. Clearly, computing all $\boldsymbol{v}_j$ using this method requires only one CMul multiplicative depth and dd'^2 CMuls, without Rotates.

TABLE I: The cost of n products of encrypted matrices of dimension $m \times d$ and plaintext matrices of dimension $d \times d'$ (#cts: the number of ciphertexts).

Packing method	#cts	#CMuls	#depths
Sequence-by-sequence	n	$n(d + dd'^2)$	2 CMul
Letter-by-letter	d	dd'^2	1 CMul

It should be noted that this method for PCMM is not new; it appeared in, e.g., [29] for CCMM. When computing n PCMMs for encrypted matrices of dimension $m \times d$ and plaintext matrices of dimension $d \times d'$ (as the query, key, value matrices in the attention layer), if we use the sequence-by-sequence packing method, we first need to extract (via Mask) the ciphertexts corresponding to the d columns from the ciphertext of the matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ of each sequence, and then perform the above PCMM. Finally, we need to repeat this process n times to complete the whole task. Table I presents the overhead of the method combining the above PCMM with the proposed letter-by-letter packing. It can be observed that, compared to the sequence-by-sequence packing method, the letter-by-letter packing method has obvious advantages.

C. CCMM

After applying the sqmax function as an approximation of softmax in Section III-C, the computation of $X^{(h)}$ in attention essentially becomes a three-matrices multiplication, where each matrix is encrypted column-wise. We now propose an optimization for this scenario. Let's first recall Halevi-Shoup's method for linear transformation on ciphertexts.

1) Halevi-Shoup's method: For a plaintext square matrix $\boldsymbol{C}$ of dimension m, the i-th *diagonal vector* of $\boldsymbol{C}$ is defined as $\mathrm{Diag}_i(\boldsymbol{C}) = (c_{0,i}, \ldots, c_{m-i-1,m-1}, c_{m-i,0}, \ldots, c_{m-1,i-1})$, denoted by $\boldsymbol{d}_i$ for simplicity. Now, for an encrypted m-dimensional vector $\boldsymbol{v}$, Halevi and Shoup in [30] utilize the following formula

$$\boldsymbol{C}\boldsymbol{v}^{\mathrm{T}} = \sum_{0\le i<m} \boldsymbol{d}_i \odot \mathsf{Rot}_i(\boldsymbol{v}). \tag{3}$$

Computing $\boldsymbol{C}\boldsymbol{v}^{\mathrm{T}}$ with this formula requires m Rots. In [31], they employ the baby-step-giant-step (BSGS) strategy, reducing m Rots to $O(\sqrt{m})$. The idea is based on the following formula (assuming $m = \ell \cdot k$)

$$\boldsymbol{C}\boldsymbol{v}^{\mathrm{T}} = \sum_{0\le i<\ell} \mathsf{Rot}_{ik}\Big(\sum_{0\le j<k} \mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}) \odot \mathsf{Rot}_j(\boldsymbol{v})\Big). \tag{4}$$

The reason BSGS can significantly reduce the number of required Rots is that the matrix $\boldsymbol{C}$ is in plaintext, allowing us to precompute $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j})$ directly in plaintext.

2) Three-matrices multiplication: Suppose that we need to compute $(\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}})\boldsymbol{V}$ with all $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V} \in \mathbb{R}^{m\times d}$ encrypted column-wise. Let $\boldsymbol{C} = \boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}} \in \mathbb{R}^{m\times m}$ and let $\boldsymbol{d}_i$ be the i-th diagonal vector of $\boldsymbol{C}$. It is well-known that the resulting ciphertexts of $\boldsymbol{C}$ are in diagonal form up to some rotations; see, e.g., [32], [33].

Let $\boldsymbol{Q} = (\boldsymbol{q}_0^{\mathrm{T}}, \ldots, \boldsymbol{q}_{d-1}^{\mathrm{T}})$ and $\boldsymbol{K} = (\boldsymbol{k}_0^{\mathrm{T}}, \ldots, \boldsymbol{k}_{d-1}^{\mathrm{T}})$, i.e., $\boldsymbol{q}_i^{\mathrm{T}}$ and $\boldsymbol{k}_i^{\mathrm{T}}$ be the i-th column of $\boldsymbol{Q}$ and $\boldsymbol{K}$, respectively. Then

$$\boldsymbol{d}_i = \sum_{0\le s<d} \boldsymbol{q}_s \odot \mathsf{Rot}_i(\boldsymbol{k}_s), \quad 0 \le i < m, \tag{5}$$

from which we can compute $\boldsymbol{C}\boldsymbol{v}^{\mathrm{T}}$ with Eq. (3) for each column $\boldsymbol{v}^{\mathrm{T}}$ of $\boldsymbol{V}$. The cost for this method is listed in Table II. Note that we cannot use the BSGS technique in this way.

3) Double BSGS: To utilize BSGS to compute $\boldsymbol{C}\boldsymbol{v}^{\mathrm{T}}$ with each column $\boldsymbol{v}^{\mathrm{T}}$ of $\boldsymbol{V}$, we assume that $m = k \cdot \ell$. Now we describe how to efficiently compute $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j})$ in encrypted form. In fact, rewriting Eq. (5) as

$$\boldsymbol{d}_{i\cdot k+j} = \sum_{0\le s<d} \boldsymbol{q}_s \odot \mathsf{Rot}_{ik+j}(\boldsymbol{k}_s), \quad (i < \ell, j < k)$$

leads to

$$\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}) = \sum_{0\le s<d} \mathsf{Rot}_{-ik}(\boldsymbol{q}_s) \odot \mathsf{Rot}_j(\boldsymbol{k}_s).$$

This indicates that we only need to rotate $\boldsymbol{k}_s$ for the *baby* steps with step size $j < k$ and rotate $\boldsymbol{q}_s$ for the *giant* steps with step size $-ik$ for $i < \ell$. This costs only $\ell + k$ rotations, and the resulting rotated vectors can be reused to construct $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j})$ for all $0 \le i < \ell$ and $0 \le j < k$. Once these rotated vectors are prepared, we can use BSGS in Eq. (4) to compute $\boldsymbol{C}\boldsymbol{v}^{\mathrm{T}}$ with another $\ell + k$ Rots. We call this technique *double BSGS (DBSGS)*.

It needs totally $O(d(k+\ell)) = O(d\sqrt{m})$ Rotates to compute $(\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}})\boldsymbol{V}$. The method without DBSGS requires $O(md)$ Rotates, and hence DBSGS saves the number of Rotates by a factor $\sqrt{m}$ at a cost of one more CMul depth; see Table II. In DASHformer, $m = 50$ so that we can set $\ell = 7$ and $k = 8$ which leads to save the number of Rotates more than 70%. For the evaluation of encrypted DASHformer, this improvement reduced the running time by more than a half; see Section VI.

TABLE II: The cost of encrypted $(\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}})\boldsymbol{V}$ w./w.o. DBSGS

DBSGS	#Muls	#CMuls	#Rotates	#depths
Without	$2md$	$4md$	$4md$	1 CMul + 2 Mul
With	$2k\ell d$	$4(k+\ell)d$	$4(k+\ell)d$	2 CMul + 2 Mul

We have two additional remarks on DBSGS: (1) The number of columns may not be limited to d since we compute

for the columns of $\boldsymbol{V}$ one by one. (2) Combining an arbitrary encoding-compatible algorithm for encrypted matrix multiplication for two matrices, e.g., [34], DBSGS can be used to compute the product of any number of matrices.

V. Optimizing Multiplicative Depth

In addition to the above techniques and algorithmic improvements, we focus on optimizing the multiplicative depth required for evaluating DASHformer to further enhance computational efficiency.

A. Multiplying by a Plaintext Diagonal Matrix

In DASHformer, we need to deal with the matrix multiplication in the form of $\boldsymbol{\Lambda XV}$, where $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda})$ is a plaintext diagonal matrix generated by the vector $\boldsymbol{\lambda} = (\lambda_i)_i$. When multiplying a CKKS encrypted ciphertext matrix by the plaintext diagonal matrix $\boldsymbol{\Lambda}$, it consumes a CMul multiplicative depth. Here, we present two methods for two cases in DASHformer to remove the need for such a CMul multiplicative depth.

1) Column encoded X*:* In this case, $\boldsymbol{X}$ is an $n \times m$ matrix encrypted in columns, and $\boldsymbol{V} = (v_{i,j})$ is in plaintext form. It happens in Eq. (1). Let $\boldsymbol{U} = \boldsymbol{\Lambda XV}$ and $\boldsymbol{x}_i^{\mathrm{T}}$ be the i-th column of $\boldsymbol{X}$. Then the j-th column of $\boldsymbol{U}$ is $\boldsymbol{u}_j^{\mathrm{T}} = \sum_{i<m}(\boldsymbol{\lambda} \odot \boldsymbol{x}_i) \cdot v_{i,j} = \sum_{i<m}((v_{i,j} \cdot \boldsymbol{\lambda}) \odot \boldsymbol{x}_i)$. This implies that one can multiply the plaintexts $v_{i,j}$ and $\boldsymbol{\lambda}$ in advance, instead of multiplying $\boldsymbol{\Lambda}$ after multiplying $\boldsymbol{X}$ and $\boldsymbol{V}$.

2) Diagonal encoded X*:* In this case, $\boldsymbol{X}$ is a square matrix of dimension n and $\boldsymbol{V}$ is encrypted in column. It happens in the layer of H-heads attention. For simplicity, we consider $\boldsymbol{\Lambda X v}$ with $\boldsymbol{v}$ a vector and $\boldsymbol{X}$ is given in the diagonal vectors $(\boldsymbol{d}_i)_i$. Now it follows from (3) that

$$\boldsymbol{\Lambda X v} = \sum_{i<n} \boldsymbol{d}_i \odot (\boldsymbol{\lambda} \odot \mathsf{Rot}_i(\boldsymbol{v}))\,,$$

from which one can combine the multiplication by $\boldsymbol{\lambda}$ with Mask within $\mathsf{Rot}_i(\boldsymbol{v})$ when the dimension $\boldsymbol{v}$ is less than $N/2$, and hence saving a multiplicative depth of CMul.

B. Further Optimizations

When evaluating DASHformer on encrypted protein sequences, if we strictly follow the workflow described in Section I and perform encrypted computations layer by layer, the required ciphertext multiplicative depth would not be less than 16, even with all previous mentioned techniques and algorithmic improvements. However, by combining certain computational steps, we can reduce the required multiplicative depth to 10. We will not display all the details here but will provide an example to illustrate how our optimization works.

Let's take the second layer normalization and the average pooling as an example. Assume that the input of $\mathsf{LayerNorm}_2$ is $\boldsymbol{X}$ and the output of the average pooling is $\boldsymbol{x}$. Then,

$$\begin{aligned}\boldsymbol{x} &= \mathbf{1} \cdot (\mathsf{LN}_2(\boldsymbol{X})\boldsymbol{W}_d + \mathbf{1}^{\mathrm{T}}\boldsymbol{b}_d) \\ &= \mathbf{1} \cdot \boldsymbol{\Sigma}_2^{-1} \cdot \boldsymbol{X} \cdot \boldsymbol{\Gamma}_2\boldsymbol{W}_d + \mathbf{1} \cdot \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{\beta}_2\boldsymbol{W}_d + \mathbf{1} \cdot \mathbf{1}^{\mathrm{T}}\boldsymbol{b}_d.\end{aligned}$$

Thanks to the optimization techniques presented in Section V-A, we can complete this computation using only one CMul multiplicative depth. If we compute it layer by layer, it would require at least three CMul multiplicative depths. [1] Table III shows that the total number of required multiplicative depths (including CMul and Mul) by our protocol is only 10, which allows us to evaluate DASHformer on multiple encrypted protein sequences without using the costly bootstrapping.

TABLE III: Multiplicative depth for encrypted DASHformer

Task	Before ReLU	ReLU	After ReLU	Total
Multiplicative depth	6	3	1	10

We call the resulting protocol *encrypted DASHformer*, whose security directly follows from the security of CKKS.

VI. Experimental Results

We implement the encrypted DASHformer using the CKKS scheme [18] in the `Lattigo` package (v5.0.2) [19] and perform plaintext matrix computations using the `gonum` package (v0.15.1) [35]. Our implementation supports multi-threads for parallel computing.

It is worth mentioning that we newly developed a CopyKeyGenerator function for `Lattigo`, which was not previously included. This function ensures that different threads can access the same key generator object, providing thread safety for multithreaded parallel computation and accelerating the process of key generation. Our implementation with all involved parameters are available at https://github.com/hangenba/enc_dashformer.

A. Setup

The parameters of CKKS are chosen as the following:

- $\log N = 14$, where N is the polynomial degree of the ring for RLWE,
- $\log Q = 38 + 33 \cdot 10 = 368$, where Q is the largest ciphertext modulus,
- $\log P = 36 \cdot 2 = 72$, where P is the special prime for key-switching,
- $\log \Delta = 33$, where Δ is the scaling factor.

1) Security: According to a draft standard for HE [36], the above setup can achieve 128-bit security.

2) Error-control: To ensure that the encrypted results (at the lowest level) still can be decrypted correctly, we scale the coefficients used during the average pooling step (see Section I. Based on the above setup and the given $1,000$ protein sequences, we multiply $\boldsymbol{W}_d$ and $\boldsymbol{b}_d$, respectively, by a constant $c = 1/2649$. Correspondingly, to make the decrypted results consistent with the plaintext results, we also multiplied the decrypted results by $1/c$.

3) Packing capability: According to the above parameter settings, each ciphertext can pack $8,192$ real numbers. Since we use the letter-by-letter tensor packing method introduced in Section IV, each ciphertext only loads 50×100 real numbers, resulting in a utilization rate of approximately 61%. If we fully

[1] We also note that the last two terms are all in plaintext form, and hence can be computed without operations on encrypted data.

utilize this packing capacity, a single encrypted DASHformer evaluation can deal with up to 163 protein sequence classifications in parallel (or, e.g., 128 protein sequences of each with 64 amino acid symbols).

B. Performance

1) Computation: We test the performance of our implementation with *four* threads on a PC (Intel i7-14700KF 3.4 GHz and 64 GB memory) with Ubuntu 24.04 (WSL). The maximum memory used during our execution is about 29 GB. The timings are given in Table IV. From Table IV, it is evident that the DBSGS technique reduces the running time by more than half, enabling us to complete the classification of 163 encrypted protein sequences within 165 seconds. On average, this amounts to about one second per sequence, which shows the near-practicality of the protocol of encrypted DASHformer. According to the report of [17, Table 2], our solution is at least 3.3x faster than other solutions of iDASH 2024-Track 1, e.g., [37].

TABLE IV: Timings in second (amortized with 163 sequences)

DBSGS	Before ReLU	ReLU	After ReLU	Total (with I/O)	Amortized
Without	343	5	3	356	2.19
With	152	5	3	165	1.02

2) Communication: Recall that each input protein sequence is converted into an $m \times l$ matrix after one-hot encoding. There are n such matrices, so the input is an $n \times m \times l$ tensor, where l is the size of the alphabet. Under the letter-by-letter encoding, this tensor is stored in l ciphertexts. After the encrypted DASHformer evaluation is completed, the classification results of these n protein sequences are stored in c ciphertexts $(\mathsf{ct}_i)_{0\le i<c}$, where c is the number of protein classes. At the end, each ct_i is decrypted to the scores of all n protein sequences being assigned to the i-th class. Therefore, for encrypted DASHformer, at the beginning of the execution, the client needs to transmit l ciphertexts to the server; after execution, the server needs to transmit c ciphertexts back to the client. For DASHformer, $l = c = 25$. In our tests, the sizes of the transmitted ciphertexts are approximately 68.77 MB and 6.26 MB, respectively. The latter one is evidently small because that the modulus size of resulting ciphertexts is only 38 while that was 368 for the input ciphertexts.

3) Accuracy: We execute our implementation of encrypted DASHformer, together with the plaintext original DASHformer and the plaintext approximated DASHformer introduced in Section III, 10 times for randomly chosen 163 out of the given protein sequences. The micro-AUCs are shown in Fig. 4. We can observe from Fig. 4 that there is a difference between the approximate DASHformer and the original DASHformer in terms of micro-AUC, but the difference is about 0.0534 on average, not significant. Because CKKS itself is an approximate homomorphic encryption scheme, the discrepancy between the encrypted DASHformer and the original DASHformer is about 0.0547 on average, a bit larger than that of the approximate one. However, fortunately, in these random tests, the micro-AUC of the encrypted DASHformer is still acceptable, ranging from 0.932 to 0.952. On the test data supplied by iDASH 2024-Track 1 for evaluation [17], the micro-AUC of the original, approximate, and encrypted DASHformer are 0.997, 0.942, and 0.941, respectively. Again, according to [17, Table 2], the best solution for micro-AUC [37] achieves 0.984, but is about 4.79x slower than our solution.

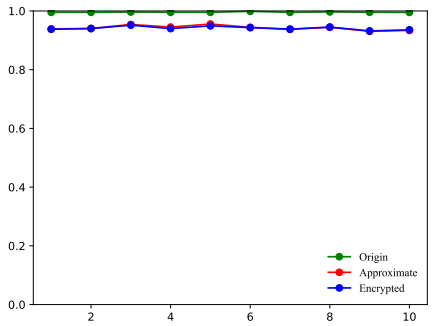


Fig. 4: Micro-AUC for randomly chosen protein sequences

VII. Related Work

Due to the increasing attention on privacy protection issues in machine learning, numerous studies on PPML have emerged recently, especially in privacy-preserving neural network inference, such as CryptoNets [38], Gazelle [39], Delphi [40], Cheetah [41], privacy-preserving CNN [42], etc. Specifically, we here only focus on privacy-preserving transformer inference (PPTI), particularly those PPTI protocols based on HE. To date, several works have been devoted to achieving efficient PPTI, including THE-X [43], Iron [44], BOLT [32], NEXUS [33], Powerformer [45], and HE-friendly LLM architectures, e.g., [46]–[48]. The challenges in PPTI lie in the secure computation with high-dimensional matrices and the secure computation of more complex nonlinear functions, e.g., softmax, LayerNorm, GELU, etc.

A. HE-Based Matrix Computation

In the aforementioned PPTI protocols, most of them use HE to perform matrix operations, primarily because some HE schemes (such as BGV, BFV, and CKKS) support SIMD operations. Therefore, how to efficiently perform homomorphically encrypted matrix computations has become a very popular research topic, roughly divided into PCMM and CCMM.

1) PCMM: There are not so many works specifically discussing the PCMM problem. Bae et al. [49] surprisingly reduces PCMM to one or two plaintext matrix-multiplication(s), but they pack data into the coefficients of plaintext polynomials rather than evaluation points. To be compatible with subsequent computations, one may have to switch between these two packing methods. This switching involves performing a linear transformation (essentially a discrete Fourier transformation) on ciphertexts with dimension N (in our setting, $N = 2^{14}$). The method presented in [50], if applied to PPTI, would involve numerous Rotates. For example, in a task of classifying n protein sequences in DASHformer, the input corresponds to an $n \times m \times d$ tensor. If the $m \times d$ encrypted

matrix on each slice needs to be multiplied with another $d \times d'$ plaintext matrix, the method in [50] would require a total number of $O(n\sqrt{d})$ Rotates. Thanks to the letter-by-letter packing method, our proposed PCMM in Section IV-B requires no Rotates and performs better than that based on the sequence-by-sequence packing method for transformer inference with multiple sequences as input.

2) CCMM: Jiang et al. [34] designed a CCMM algorithm by generalizing the matrix diagonalization encoding method used by Halevi and Shoup [30], [31], which is now commonly adopted in PPTI protocols. Subsequently, many other CCMM algorithms have been proposed; see, e.g., [51] and references therein. However, in PPTI, CCMM mainly appears in the computation of multi-head attention, which involves multiple consecutive multiplications of encrypted matrices. Directly applying existing CCMM techniques is not the optimal choice. We proposed the DBSGS technique for the consecutive multiplication of three encrypted matrices, which can significantly reduce the computational overhead. Compared to methods that do not use this technique, in the evaluation of encrypted DASHformer, we save about 70% of Rots and more than 50% of total running time.

We note that even in the state-of-the-art solutions for HE-based PPTI like BOLT [32] and NEXUS [33], the column packing method was adopted, however, the DBSGS technique was not used. (One possible reason is that the packing method is incompatible with the computation of the activation functions, like softmax.) Hence, DBSGS may be used to further speedup existing PPTI protocols.

B. Secure Computation of Activation Functions

Recently, Cho et al. [52] discussed how to evaluate the softmax function on HE-encrypted data. In existing PPTI protocols, most of them utilize MPC to compute nonlinear activation functions, which leads to extensive interactions among participants. Zimerman et al. [46] were the first to propose replacing transformer models with HE-friendly structures to achieve non-interactive inference. NEXUS [33] and Powerformer [45] propose specific modification methods. Rho et al. [48] even incorporate fine-tune into the PPTI framework. The basic idea of these methods is to use HE-friendly approximate functions to replace various nonlinear activation functions appeared in transformers. Along with this line, we present our method for DASHformer in Section III. In particular, we propose to learn the parameters for these approximate functions from the given data, fit the optimal values of these parameters, and apply them to the evaluation of encrypted DASHformer. This process seems similar to fine-tune with the given data, achieves the goal of privacy-preserving, and ensures the accuracy of inference.

VIII. Conclusion

We propose encrypted DASHformer, a non-interactive, efficient and accurate PPTI protocol for protein sequence classification based on CKKS without bootstrapping. It is arguably the first protocol that is capable of dealing with multiple input sequences in parallel within one execution of the protocol. For that, we designed a series of new techniques and algorithmic improvements to compute the non-polynomial functions, pack data, and multiply matrices, which may be useful for more HE-based applications. The experimental results show its near-practicality.

Acknowledgment

We would like to thank the organizers of iDASH 2024-Track 1 for their efforts in organizing this competition and preparing the DASHformer model and the data of protein sequences.

Conflict of Interest

Lizhong Dai is the chairman of Sansure Biotech, Inc. This affiliation had no influence on the design, data analysis, or the results and interpretation of this research. The authors declare no other potential conflicts of interest.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[2] D. Ofer, N. Brandes, and M. Linial, "The language of proteins: NLP, machine learning & protein sequences," *Computational and Structural Biotechnology Journal*, vol. 19, pp. 1750–1758, 2021.

[3] T. Idhaya, A. Suruliandi, and S. P. Raja, "A comprehensive review on machine learning techniques for protein family prediction," *The Protein Journal*, vol. 43, no. 2, pp. 171–186, 2024.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017, Long Beach, USA)*, ser. Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, vol. 30.

[5] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018, https://openai.com/research/language-unsupervised.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Minneapolis, USA, June 2 –7, 2019)*, J. Burstein, C. Doran, and T. Solorio, Eds. ACL, 2019, pp. 4171–4186.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, https://arxiv.org/abs/2302.13971.

[8] N. Brandes, D. Ofer, Y. Peleg, N. Rappoport, and M. Linial, "ProteinBERT: a universal deep-learning model of protein sequence and function," *Bioinformatics*, vol. 38, no. 8, pp. 2102–2110, 2022.

[9] Z. Du, X. Ding, Y. Xu, and Y. Li, "UniDL4BioPep: a universal deep learning architecture for binary classification in peptide bioactivity," *Briefings in Bioinformatics*, vol. 24, no. 3, p. bbad135, 2023.

[10] S. McCallum, "ChatGPT banned in Italy over privacy concerns," 2023, https://www.bbc.com/news/technology-65139406.

[11] A. Mok, "Amazon, Apple, and 12 other major companies that have restricted employees from using ChatGPT," 2023, https://www.businessinsider.com/chatgpt-companies-issued-bans-restrictions-openai-ai-amazon-apple-2023-7.

[12] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (Nov. 3–5, 1982, Chicago, USA)*, N. Pippenger, Ed. Los Alamitos: IEEE Computer Society, 1982, pp. 160–164.

[13] C. Dwork, "Differential privacy," in *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Heidelberg: Springer, 2006, vol. 4052, pp. 1–12.

[14] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing (May 31 - June 2, 2009, Bethesda, USA)*, M. Mitzenmacher, Ed. New York: ACM, 2009, pp. 169–178.

[15] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. IEEE, 2015, pp. 57–64.

[16] iDASH, "iDASH Secure Genome Analysis Competition," http://www.humangenomeprivacy.org, Accessed on Nov. 2024.

[17] A. Harmanci, L. Chen, M. Kim, and X. Jiang, "Descriptor: Benchmarking secure neural network evaluation methods for protein sequence classification (iDASH24)," *IEEE Data Descriptions*, 2024, https://doi.org/10.1109/IEEEDATA.2024.3482283.

[18] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proceedings of ASIACRYPT 2017 – 23rd International Conference on the Theory and Applications of Cryptology and Information Security (December 3-7, 2017, Hong Kong, China), Part I*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds. Heidelberg: Springer, 2017, vol. 10624, pp. 409–437.

[19] EPFL-LDS and Tune-Insight, "Lattigo v5.0.2," Online: https://github.com/tuneinsight/lattigo, Nov. 2023.

[20] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc 3rd ITCS (January 8–10, 2012, Cambridge, MA, USA)*, S. Goldwasser, Ed. New York: ACM, 2012, pp. 309–325.

[21] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology – Proc CRYPTO 2012 (August 19–23, 2012, Santa Barbara, CA, USA)*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds. Heidelberg: Springer, 2012, vol. 7417, pp. 868–886.

[22] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive https://eprint.iacr.org/2012/144, 2012.

[23] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of ACM*, vol. 60, no. 6, pp. 43:1–35, 2013.

[24] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *Proceedings of the 15th International Conference on Availability, Reliability and Security (Virtual Event, Ireland, August 25 - 28, 2020)*, M. Volkamer and C. Wressnegger, Eds. New York: ACM, 2020, pp. 14:1–10.

[25] L. Jiang and L. Ju, "FHEBench: Benchmarking fully homomorphic encryption schemes," 2022, https://doi.org/10.48550/arXiv.2203.00728.

[26] O. Goldreich, *Foundations of Cryptography – Volume II Basic Applications*. Cambridge: Cambridge University Press, 2004.

[27] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*, ser. NATO ASI Series, F. F. Soulié and J. Hérault, Eds. Heidelberg: Springer, 1990, vol. 68, pp. 227–236.

[28] A. Wuraola, N. Patel, and S. K. Nguang, "Efficient activation functions for embedded inference engines," *Neurocomputing*, vol. 442, pp. 73–88, 2021.

[29] W.-j. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," in *NDSS 2017: 24th Annual Network and Distributed System Security Symposium (San Diego, USA, February 26–March 1, 2017)*. The Internet Society, 2017.

[30] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology – CRYPTO 2014 (Santa Barbara, USA, August 17-21, 2014)*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds. Heidelberg: Springer, 2014, vol. 8616, pp. 554–571.

[31] ——, "Bootstrapping for HElib," in *Advances in Cryptology – Proc EUROCRYPT 2015 (April 26–30, 2015, Sofia, Bulgaria), Part I*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds. Heidelberg: Springer, 2015, vol. 9056, pp. 641–670.

[32] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *2024 IEEE Symposium on Security and Privacy (May 19–May 23 2024, San Francisco, USA)*. Los Alamitos, CA, USA: IEEE Computer Society, 2024, pp. 133–133.

[33] J. Zhang, X. Yang, L. He, K. Chen, W. jie Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," in *NDSS '25*, 2025, https://eprint.iacr.org/2024/136.

[34] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (October 15–19, 2018, Toronto, Canada)*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. New York: ACM, 2018, pp. 1209–1222.

[35] The gonum team, "gonum v0.15.1," Online: https://https://github.com/gonum/gonum, August 2024.

[36] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018, https://eprint.iacr.org/2019/939.pdf.

[37] J.-P. Bossuat, "A solution for iDASH 2024 - HE track," 2024, https://github.com/gausslabs/idash-2024-solution.

[38] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *The 33rd ICML*, ser. PMLR, M. F. Balcan and K. Q. Weinberger, Eds. New York: PMLR, 2016, vol. 48, pp. 201–210.

[39] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proceedings of The 27th USENIX Security Symposium (August 15-17, 2018, Baltimore, MD)*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 1651–166.

[40] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proceedings of the 29th USENIX Security Symposium, USENIX Security 2020 (August 12-14, 2020, Virtually)*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 2505–2522.

[41] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *USENIX Security '22 (August 10–12, 2022, Boston, USA)*. Boston: USENIX Association, 2022, pp. 809–826.

[42] D. Kim and C. Guyot, "Optimized privacy-preserving CNN inference with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2175–2187, 2023.

[43] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, "THE-X: Privacy-preserving transformer inference with homomorphic encryption," in *Findings of the Association for Computational Linguistics: ACL 2022 (Dublin, Ireland, May 22-27, 2022)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Stroudsburg: ACL, 2022, pp. 3510–3520.

[44] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," in *36th Conference on Neural Information Processing Systems (NeurIPS 2022, November 28–December 9, 2022)*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds. Curran Associates, Inc., 2022, pp. 15 718–15 731.

[45] D. Park, E. Lee, and J.-W. Lee, "Powerformer: Efficient privacy-preserving transformer with batch rectifier-power max function and optimized homomorphic attention," Cryptology ePrint Archive, 2024/1429, 2024, https://eprint.iacr.org/2024/1429.

[46] I. Zimerman, M. Baruch, N. Drucker, G. Ezov, O. Soceanu, and L. Wolf, "Converting transformers to polynomial form for secure inference over homomorphic encryption," 2023. [Online]. Available: https://arxiv.org/abs/2311.08610

[47] X. Liu and Z. Liu, "LLMs can understand encrypted prompt: Towards privacy-computing friendly transformers," 2023. [Online]. Available: https://arxiv.org/abs/2305.18396

[48] D. Rho, T. Kim, M. Park, J. W. Kim, H. Chae, J. H. Cheon, and E. K. Ryu, "Encryption-friendly LLM architecture," 2024. [Online]. Available: https://arxiv.org/abs/2410.02486

[49] Y. Bae, J. H. Cheon, G. Hanrot, J. H. Park, and D. Stehlé, "Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused," in *Advances in Cryptology – CRYPTO 2024*, ser. Lecture Notes in Computer Science, L. Reyzin and D. Stebila, Eds. Cham: Springer, 2024, vol. 14922, pp. 387–421.

[50] Y. Liu, L. Yang, J. Chen, W. Wu, and Y. Feng, "Matrix computation over homomorphic plaintext-ciphertext and its application," *Journal on Communications*, vol. 45, no. 2, pp. 150–161, 2024, in Chinese.

[51] J. Chen, L. Yang, W. Wu, Y. Liu, and Y. Feng, "Homomorphic matrix operations under bicyclic encoding," *IEEE Transactions on Information Forensics & Security*, 2024, https://doi.org/10.1109/TIFS.2024.3490862.

[52] W. Cho, G. Hanrot, T. Kim, and D. S. Minje Park, "Fast and accurate homomorphic softmax evaluation," in *CCS '24 (Salt Lake City, USA)*. New York: ACM, 2024.