CryptoGCN: Fast and Scalable Homomorphically Encrypted Graph Convolutional Network Inference

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recently cloud-based graph convolutional network (GCN) has demonstrated great success and potential in many privacy-sensitive applications such as personal healthcare and financial systems. Despite its high inference accuracy and performance on cloud, maintaining data privacy in GCN inference, which is of paramount importance to these practical applications, remains largely unexplored. In this paper, we take an initial attempt towards this and develop *CryptoGCN*–a homomorphic encryption (HE) based GCN inference framework. A key to the success of our approach is to reduce the tremendous computational overhead for HE operations, which can be orders of magnitude higher than its counterparts in the plaintext space. To this end, we develop an approach that can effectively take advantage of the sparsity of matrix operations in GCN inference to significantly reduce the computational overhead. Specifically, we propose a novel AMA data formatting method and associated spatial convolution methods, which can exploit the complex graph structure and perform efficient matrix-matrix multiplication in HE computation and thus greatly reduce the HE operations. We also develop a co-optimization framework that can explore the trade offs among the accuracy, security level, and computational overhead by judicious pruning and polynomial approximation of activation module in GCNs. Based on the NTU-XVIEW skeleton joint dataset, i.e., the largest dataset evaluated homomorphically by far as we are aware of, our experimental results demonstrate that *CryptoGCN* outperforms state-of-the-art solutions in terms of the latency and number of homomorphic operations, i.e., achieving as much as a 3.10$\times$ speedup on latency and reduces 77.4% of total Homomorphic Operation Count with a small accuracy loss of 1-1.5$\%$.

1 Introduction

Graph Convolutional Neural Networks (GCNs) have recently emerged in machine learning and demonstrated superior performance in various privacy-sensitive applications such as human action recognition, financial recommendation system, and autonomous driving. While it has been increasingly popular to deploy machine learning services on cloud, the cloud environment raises critical concerns for GCN-based privacy-sensitive services, since graph data usually contain a considerable amount of sensitive information.

Recently, the Homomorphic Encryption (HE) based private-preserving machine learning (PPML) has emerged to be an effective way to protect the privacy of clients. Using HE, a client can encrypt the data and send it to cloud. The cloud server directly operates on the encrypted data and sends the encrypted results back to the client, which can then be decrypted and used. As the data are encrypted throughout the entire process when it is out of the client control, the data privacy is greatly enhanced. The challenge, however, is to deal with the tremendously increased computational cost associated

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

with the HE operations (e.g., multiplications, additions), which is orders of magnitude higher than that in the plaintext space [4, 12].

Recent work [1, 3, 4, 8, 9] that employ HE to build privacy-preserving inference framework are all focused on the traditional Convolutional Neural Network (CNN) models. However, these approaches become ineffective when they are directly applied for the implementation of GCN. HE computation not only increases the computational overhead tremendously, but also exhibit some unique features than its counterpart in the plaintext space. Current HE-based work are normally built on so-called leveled HE scheme, which has a limit on the maximum number of concatenated homomorphic multiplication operations. While GCN has the advantage of capturing the spatial and temporal relationship of different graph node, GCN inferencing demands extra matrix multiplication operations, which significantly increase not only the numbers of the HE operations but also the depth of the computation circuit. As shown in Figure 1a, the additional matrix multiplications in GCN can significantly increase the required HE operations. For a typcial 64-channel-GCN layer with matrix size 25×25, the matrix multiplication can lead to nearly $49\times$ increase in terms of homomorphic operation count (HOC) in the worst case. How to deal with the significantly increased HE operations in GCN inferencing process presents a primary challenge in our approach. In addition, as shown in Figure 1b, the choice of parameters for HE schemes not only affects the security level of the computation, but also has profound impacts to the numbers of HE operations. How to judiciously choose the HE parameter to optimize the security, computational cost, and latency is also critical for effective and efficient GCN inferencing.

In this paper, we have made the following contributions. First, we develop an approach that can effectively take advantage of the sparsity of matrix operations in GCN inferencing that can significantly reduce the computational overhead. As shown above, for GCN inferencing, the required matrix-matrix multiplication can lead to significantly increased HE operations. In the meantime, the matrix operation for GCN inferencing exhibits strong sparsity features, which can be exploited to reduce the computational overhead. To this end, we develop a novel GCN data formatting method, i.e., Adjacency Matrix Aware data formatting method, and associated spatial/temporal convolutional algorithms, which can exploit the single instruction multiple data (SIMD) structures in HE computation and thus greatly reduce the HE operations. Second, we also study how to better manage the HE computation numbers and levels for GCN inferencing by judiciously pruning and approximation of the activation module in GCN and settings of HE parameters. We develop a co-optimization framework that can help to explore the tradeoffs among security level, inferencing accuracy, and inferrencing latency. Third, we have conducted extensive experiments based on the NTU-XVIEW skeleton joint dataset. Our experimental results show that the AMA data formatting achieves a latency speedup of up to $3.10\times$, the Activation Prune achieves as much as $2.29\times$ speedup for latency. To our best knowledge, this is the first work that builds the PPML pipeline for GCN-based models with HOC decrease by as much as 77.44% compared to previous benchmarks.

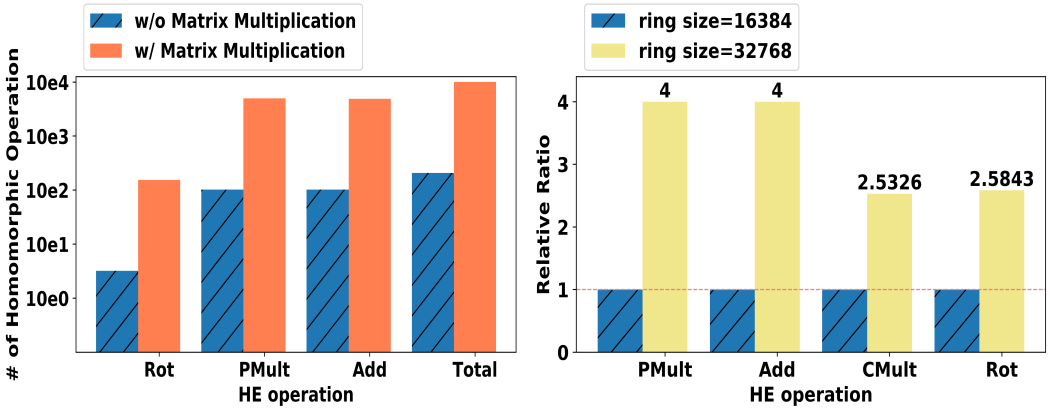
2 Background and Related Work

Spatial-Temporal Graph Convolution Network. The Spatial-Temporal Graph Convolution Network (ST-GCN) [16] contains two types of convolutions, spatial and temporal convolution, designed to extract spatial and temporal information from the input graph data respectively. The spatial convolution is described by Equation 1.

$$H = \sum_j \tilde{D_j}^{-\frac{1}{2}} \tilde{A}_j \tilde{D_j}^{-\frac{1}{2}} X W_j \tag{1}$$

in which j indicates different sets of graph connections separated by a partition strategy to better extract the spatial information. X is the input graph data. $W_j \in \mathbb{R}^{C_{in} \times C_{out}}$ is a set of filter parameters to transform the input tensor from channel C_{in} to channel C_{out}. $\tilde{D_j}$ is the degree matrix. $\tilde{A}_j$ is the adjacency matrix with self-loop. The XW_j is implemented by a 2D-convolution with kernel size 1×1, then multiplied with the normalized adjacency matrix $\tilde{D_j}^{-\frac{1}{2}} \tilde{A}_j \tilde{D_j}^{-\frac{1}{2}}$, and then the resulted data is summed up. The Temporal Convolution is a convolution with the same graph node data from different time slices. Thus, it can extract information on different graph nodes from temporal domain.

CKKS Homomorphic Encryption Scheme. The CKKS scheme [2], which is based on the hardness of ring learning with errors (RLWE) problem, is an leveled homomorphic encryption scheme that



(a) (b)

Figure 1: Motivation examples: (a) the number of HE operations increased at log scale (by $\sim 49\times$ in total) due to involving an adjacent matrix-based matrix multiplication in a typical 64-channel GCN layer (detailed setting in Sec. 4); (b) the number of HE operations on a single ciphertext increased by up to $4\times$ (32K v.s. 16K, normalized to 16K) due to enlarging the ring size. *PMult*, *Add*, *CMult* and *Rot* represent ciphertext-plaintext multiplication, ciphertext addition, ciphertext-ciphertext multipication and rotation, respectively (see Sec. 2).

allows arithmetic operations on encrypted data over real and floating point numbers. CKKS provides configurable precision by taking the encryption noise as natural error introduced in the approximation computations and through dropping the least significant bits of computations via the rescaling of ciphertext. The supported homomorphic operations includes the ciphertext addition *Add*(ct_1, ct_2), ciphertext multiplication *CMult*(ct_1, ct_2), scalar multiplication *PMult(ct , pt)*, rotation *Rot(ct , k)* and rescaling *Rescale(ct)*. The scalar multiplication is to multiply a ciphertext with plaintext. The rotation is to apply Galios automorphisms of the cyclotomic extension to the plaintext polynomials in encrypted form resulting in a cyclic shift of the slot vector. For instance, $Rot(ct, k)$ transforms an encryption of $(v_0, ..., v_{N/2-1})$ into an encryption of $(v_k, ..., v_{N/2-1}, v_0, ..., v_{k-1})$.

Compared with unencrypted computations, CKKS introduces a significant runtime and memory overhead. The use of packing, also referred to as batching, allows to slot multiple data values into one ciphertext so the encrypted computations can be done in a Single Instruction Multiple Data (SIMD) manner. We use this property to improve the amortized overhead in this paper.

The security of the CKKS encryption scheme is measured in bits. With $\lambda = 128$, it will take around 2^{128} operations to break the encryption. Throughout this paper, we assume that N is the cyclotomic polynomial degree. The CKKS is a leveled HE. The level of a ciphertext (l) is defined as the number of successive multiplications that can be performed to the ciphertext without bootstrapping. The level is decreased by one through the rescaling operation after each homomorphic multiplication. If the level becomes zero, the bootstrapping is needed to make this zero-level ciphertext a higher-level ciphertext to enable further homomorphic operations. In our work, we optimize the GCN network to have lower depth and select proper parameters to avoid the costly bootstrapping procedure.

Related Work. CryptoNets [4] is the first work that demonstrates the feasibility of building PPML by HE. CryproNets evaluates a 5-layer NN with MNIST dataset and can perform inference on 59000 pictures within one hour. However, the long inference latency makes it hard to be applied to large-scale models and datasets. A more recent work, SHE [9], preserves and translates ReLU and maxpooling operations by Boolean operations and implemented by TFHE, and achieved the state-of-the-art inference accuracy. However, SHE also has high inference latency issues, where making a prediction on a CIFAR-10 size image requires 2258 seconds. Some MPC [13, 7, 11] combined with HE framework offer low inference latency. However, they suffer from high communication overhead in terms of data transfer. For example, DeepSecure [13] needs to exchange 722GB of data between the client and the server for only a 5-layer CNN inference on one MNIST image. The studies by LoLa [1], CHET [3], and HEAR [8] target to use the natural ciphertext packing technique to place multiple value from network nodes in the same ciphertext to perform HE operation in a single instruction multiple data (SIMD) way. However, in their frameworks, the data representation in ciphertext is not

optimized for GCN-based models so a large amount to HE operations (multiplication and addition) are needed.

3 Methodology

In this section, we first present the threat model assumption for this work. We then discuss technique details of our proposed *CryptoGCN*–a CKKS-based homomorphic encryption framework tailored for fast and scalable GCN inference on encrypted graph tensor. In particular, we focus on two key components to significantly reduce the number of HE operations: 1) adjacency matrix-aware (AMA) data formatting dedicated to simplifying GCN's matrix-matrix multiplications without involving ciphertext rotation; 2) non-linear activation pruning to reduce the multiplicative depth, resulting in trade-offs between security level and inference latency with low accuracy loss.

Threat Model. We assume the cloud-based machine learning service, of which a well-trained graph convolutional network model with plaintext weights, is hosted in a cloud server. A client can upload the private and sensitive data to cloud for obtaining the online inference service. The cloud server is semi-honest (e.g. honest but curious). To ensue the confidentiality of clients' data against such a cloud server, a client will encrypt the data by HE and send it to cloud for performing encrypted inference without decrypting the data or accessing client's private key. The client then can decrypt the returned encrypted inference outcome from cloud using the private key. In this work, we focus on encrypting graph node features and the normalized adjacency matrix (often sparse and same for different graph inputs) is assumed as plaintext.

3.1 AMA Data Formatting and Matrix-Matrix Multiplication

Since HE operations can be performed on encrypted vectors by taking advantage of the SIMD architecture for parallel computing, the input tensor should be well placed into a "big vector container" by a certain format which we call "data formatting". The state-of-the-art row-major format [3, 8] concatenates a ciphertext row by row and facilitates the dot-product computation that is essential to convolution or matrix-based operations. However, as we shall show later, it cannot efficiently support GCNs' matrix-matrix multiplication that would occur multiple times in a multi-channel GCN layer, because of the extensive ciphertext rotation, addition and multiplication. The mini-batch inference over multiple GCN layers would further escalate HE overhead and latency for the row-major format.

Adjacency Matrix-Aware (AMA) Data Formatting. We use the skeleton-based action recognition graph tensor data as an example to better illustrate our proposed AMA data formatting: assuming the size of an input graph tensor is $B \times C \times T \times J$, where B is the batch size of mini-batch inference (e.g. $B = 1$ for a single input inference), C, T and J are the number of input channels, video frames and joints, respectively. The first step is to permute the tensor as J columns, with each column of size $C \times B \times T$. Then each column is flattened to C 1D vectors, with each vector of size $B \times T$. For each of such 1D vector, zeros are padded to make its length equal to the smallest power-of-two integer greater than $B \times T$. The C zero-padded 1D vectors are concatenated and then further stacked to a plaintext vector via a channel-interleaving manner until the space of a plaintext vector can be fully exploited, e.g. the length reaches half of the ring size N. Finally, we encrypt such a plaintext as a fully-packed ciphertext ct_k, $k \in J$. The detailed process is presented in Algorithm 1 and Fig. 2(a).

AMA Data Format-Aided Matrix-Matrix Multiplication. Once the AMA data formatted ciphertext is created, the next step would be to apply it to simplify and speedup the matrix-matrix multiplication introduced by adjacency matrix and convolution operations. Recall Eq. 1 in Section 2, a typical ST-GCN layer's computation involves three consecutive *PMult(ct , pt)*: spatial convolution (1×1 plaintext kernel), matrix-matrix multiplication with normalized adjacency matrix $J \times J$ in plaintext, and the temporal convolution along the dimension T ($K \times 1$ plaintext kernel). Considering the property of 1×1 spatial convolutional kernel, we can easily merge this into the adjacency matrix and formulate a new plaintext matrix to reduce one multiplicative depth of $PMult$. Then we can perform the matrix-matrix multiplication based on the new $J \times J$ plaintext matrix A. Note that each graph tensor input channel could contain one such matrix.

For row-major formatted ciphertext, as Fig. 2(c) shows (bottom), even with state-of-the-art diagonal encoding method [5], such matrix-matrix multiplication would still involve $2J - 1$ ciphertext rotations (*Rot*). Since the final outcome is the sum of each rotated ciphertext (*Rot(ct,k)*) multiplied by the

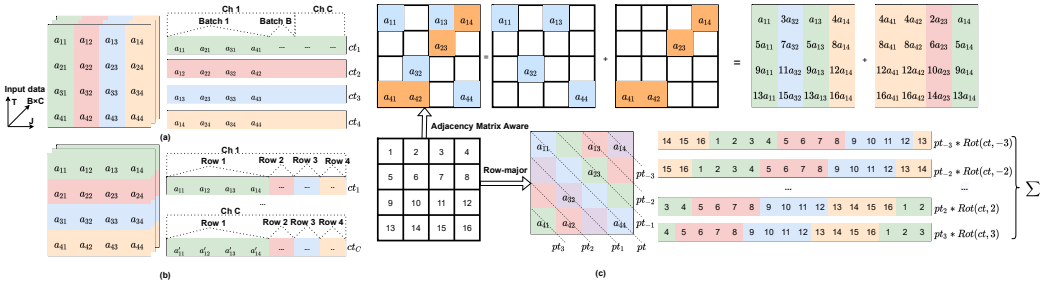


Figure 2: The comparison of data formatting: (a) AMA data format; (b)Row-major format; (c) Matrix-matrix multiplication comparison using AMA and row-major formatted ciphertext.

corresponding diagonal encoded vector (Pt_k) from the plaintext matrix A, it also brings extra *PMult* and *Add*. In contrast to the row-major data format, our AMA data format can significantly reduce the amount of these HE operations. As Fig. 2(c) (top) demonstrates, first, we decompose the $J \times J$ plaintext matrix A into a series of patterned sparse matrices A_i–each A_i contains at most one valid element in each column, and $A = \sum_{i=1}^{m} A_i, m \leq J$. Second, we simply multiply the column-wise fully-packed ciphertext (due to the AMA data formatting) with the valid element of the corresponding column in A_i, and then sum the m intermediate column-wise ciphertext to obtain a final ciphertext:

$$ct'_k = \sum_{i=1}^{m} ct_k A_i = \sum_{i=1}^{m} \sum_{k=1}^{J} PMult(ct_{i_k}, a_{i_k k}) \tag{2}$$

Where $a_{i_k k}$ represents the single valid element in column k of A_i. Since the process does not require any *Rot*, except the simple column-wise *PMult* with a single plaintext value and final summation, the HE operations can be greatly decreased comparing with the row-major format.

Theoretical Analysis of HE Operation Reduction. We analytically compare the number of HE operations needed for matrix-matrix multiplication between our AMA and row-major data formats. To ensure the evaluation generality, our analysis is conducted by assuming a mini-batch inference of multi-channel graph tensor input ($B \times C \times T \times J$) on a typical ST-GCN layer which consists of C input channels and C output channels with 1×1 convolution kernels. The $J \times J$ matrix A in Eq. 2 can be decomposed into J sub-matrices if A is dense in the worst case. For a fair comparison, we assume the space of a ciphertex–$T \times J$ is fully exploited in both methods. As shown in Fig. 2 (a) and (b), a ciphertext in row-major and AMA- format contains one channel data of size $T \times J$, and multiple-channel multiple-batch data of size $B \times T \times c$ ($c = J/B, c \in \mathbb{N}^+$), respectively. This results in the same total amount of ciphertext–$B \times C$ in both methods. The implementations of an example matrix-matrix multiplication using the two methods are presented in Fig. 2 (c).

Since AMA formatting increases processing parallelism in SIMD by packing multiple-channel and multi-batch data into a ciphertext, it involve the rotation of the outcomes of matrix-matrix multiplications from different channels and then sum up them to obtain the result for an output channel (Fig. 2 (b)). A ciphertext including data from more channels (a larger C) requires more rotations. To further reduce the rotation overhead, we leverage the multi-channel convolutional technique and baby-step strategy from HEAR [8][1]. We also apply them to row-major formatting. However, the improvement is limited since a row-major formatted ciphertext only can contain a single channel data.

Table 1 reports the detailed breakdown of HE operation numbers for each method (see Appendix A for detailed proof). Overall, AMA data format requires almost half of *PMult* and *Add* operations of that of row-major format, since row-major format needs to compute $2J - 1$ versions of a ciphertext due to rotations in the matrix-matrix multiplication. The number of rotations increases proportionally as the number of channels C that can be included in a ciphertext increases for both methods. However, it is much less than that of row major. The rotation number difference between AMA and row-major format can be further enlarged when only the batch size B increases. This is because the AMA formatted ciphertext contains data from more batches, resulting in the reduction of rotation times ($JC - BC$). In contrast, the rotation amount of the row-major format grows as B increases. This

[1]We optimize the $K \times 1$ temporal convolution following the matrix-matrix multiplication in a similar manner.

Table 1: Analytical comparison of HE Operation # (AMA v.s. Row-major)

HE Operation	Description	Total	Complexity
Rot	Row-major	$B \times C \times (2J-2)$	$O(B \cdot C \cdot 2J)$
Rot	AMA	$B \times C \times (J/B-1)$	$O(C \cdot (J-B))$
PMult	Row-major	$B \times C \times (2J-1) \times C$	$O(B \cdot C^2 \cdot 2J)$
PMult	AMA	$J \times J \times (BC/J) \times C$	$O(B \cdot C^2 \cdot J)$
Add	Row-major	$B \times C \times (2J-1) \times C - BC$	$O(B \cdot C^2 \cdot 2J)$
Add	AMA	$J \times J \times (BC/J) \times C - BC$	$O(B \cdot C^2 \cdot J)$

means that our AMA data formatting performs better when the batch size increases. Moreover, since A is often a sparse matrix in practical ST-GCNs, the number of decomposed matrices m can be much smaller than J for matrix-matrix multiplication, indicating better efficiency than Table 1 which is obtained from a dense matrix. We further validate these observations in Section 5.

Algorithm 1 AMA Formatting for Encryption

Input: $X_k = (x_1, x_2, \ldots, x_C) \in \mathbb{R}^{C \times B \times T}$
Output: ct_k, a fully packed ciphertext

For i **from** 1 **to** C
  $v_i =$ *Flatten* x_i
  $Pad(v_i) =$ *Padding* v_i *to nearest power of 2*
end for
$V_k = (Pad(v_1), Pad(v_2), \ldots, Pad(v_C))$
$V_{kfull} =$ *stack copies of* V_k *to size* $N/2$
return $ct_k = Encrypt(V_{kfull})$

Algorithm 2 Activation Prune

Input: NN Model with M **activation layers**
Output: Optimized Architecture
For i **from** 1 **to** M
  acc_i**=Train the Model with** i_{th} **activation layer pruned**
$rank_{act} = [acc_i]$
SORT $rank_{act}$ **from high to low**
For i **from** 1 **to** M
  Fine-tune the network $Arch_i$ **without top i activations in** $rank_{act}$
  AP-performance $= (Acc_i, Level_i)$
return $Arch_i$ *with best AP-performance*

3.2 Activation Pruning

Applying the Leveled Homomorphic Encryption scheme like CKKS for private inference of a deep network is challenging because of the limited modulus bits for the encrypted ciphertext. This means that one LHE encrypted ciphertext has a limited rescale level. For computation like batch normalization, it could be easily absorbed in the adjacent linear computation like convolution and matrix multiplication [4] since batch normalization is a fixed-parameter based linear transformation in inference. However, the nolinear ReLU activation computation must be replaced by polynomial approximate activation (PAA). However, even a simple degree-2 PAA would consume 2 rescale levels. Given that each GCN layer can contain one or more ReLu activation for accuracy purpose, more rescale levels would be needed for deep GCNs. This increases the polynomial degree to achieve a certain security level, leading to higher computation overhead and inference latency.

To achieve the trade-off between inference latency, security level and accuracy, we investigated the state-of-the-art solution–fine-grained channel-wise PAA for ReLU in SAFTNet [10], which is to replace the ReLU activation in the same layer with different polynomial degrees. However, the method has a major limitation when applied to LHE-based private inference. Given a ciphertext could contain data from multiple channels in our AMA data formatting, a mask vector which choose different PAAs much be utilized in order to process a ciphertext. As a result, this would consume an extra rescale level and offset the benefit of reduced multipicative level in ReLu. Therefore, we propose to identify and prune ReLu activations based on the fact that removing ReLu activation of some layers in deep networks leads to marginal accuracy loss [6]. We name such a technique as Activation Pruning (AP). As Algorithm 2 shows, we first replace all the Relu activations with a 2 degree polynomial activation $ax^2 + bx + c$, in which (a, b, c) is updated during the training process, and train this model's accuracy to a desired level as the baseline. Second, we rank the activation layers according to the accuracy, search the architecture by increasing the number of pruned activation layers and fine-tune the new model to obtain the accuracy. A model can be selected based on the accuracy and number of needed rescale levels to trade-off between security level and latency.

Table 2: Model Architecture

Model	Layer	ST-GCN 1	ST-GCN 2	ST-GCN 3
64-STGCN-3	Output featuremap size Channels	(256,25) 64	(128,25) 128	(128,25) 128
128-STGCN-3	Output featuremap size Channels	(256,25) 128	(128,25) 256	(128,25) 256

4 Experiment Methodology

HE parameter setting. We have two sets of encryption parameters for experiment without Activation Prune and with Activation Prune. For both settings, we choose the scaling factor $\Delta = 2^{33}$ to maintain the inference accuracy. For each level it will consume 33 bits of ciphertext modulus Q. For experiment without AP setting, it requires 21 levels for the whole network architecture. Thus, we set $Q = 740$, and the polynomial degree N should be set to 2^{15} to make the security level ≥ 80 bits. For experiment with AP setting, the total level is 19 or 17. Therefore, we set $Q = 680$ *or* 600 and the polynomial degree $N = 2^{14}$ to achieve at least 80-bit security level.

Dataset. NTU-RGB+D [15] is the current largest dataset with 3D joints annotations for human action recognition task. It contain 56,880 action clips in 60 action classes. The annotations contain the 3D information (X, Y, Z) of 25 joints for each subject in the skeleton sequences. We choose one benchmark **NTU-cross-View** (NTU-XView) as the dataset for our evaluation because this benchmark is a representative human skeleton joints dataset. It contain 37,920 and 18,960 clips for training and evaluation, respectively. For better evaluation, we use 256 frames from the video clip as our input data. Thus, the input tensor size $2 \times 3 \times 256 \times 25$ contain the 25 skeleton-joint information (X, Y, Z) for 2 person in a video has 256 frames.

Network Architecture. ST-GCN [16] is the state-of-the-art GCN architecture in human action recognition task, which combines the GCN and CNN to better extract the spatial and temporal information than the previous models. In our experiments, we use a stack of 3 ST-GCN layers with a global average pooling layer and a fully-connected layer and study one small net and one large net: 64-STGCN-3 and 128-STGCN-3, where the first number stands for the channel number of the first ST-GCN layer. Table 2 summarizes the network architecture. One ST-GCN layer is composed of one Spatial Conv layer and one Temporal Conv layer. Following the stacked 3 ST-GCN layers are a global average pooling layer and a fully-connected layer. We use Stochastic Gradient Descent (SGD) optimizer with a mini-batch size of 64, a momentum of 0.9, and a weight decay of e^{-4} to train the model for 200 epochs. The initial learning rate was set to 0.01 with a decay of 0.1.

Evaluation Setup. Our experiments are conducted on a machine with AMD Ryzen Threadripper PRO 3975WX with a single thread setting. We are using Microsoft SEAL version 3.7.2 [14].

To perform the temporal convolution, we leverage the baby-step strategy [8] to do multi-channel convolution. Global Avg Pooling layer and fully-connected layer have a small impact on total latency so that we just apply straightforward approach to compute these two layers.

5 Results and Analysis

5.1 Activation Prune Ablation Study

We first analyze the effect our Activation Prune (AP) technique in our framework by a ablation study. To evaluate the performance, we optimize our neural network architecture by pruning the activation layers. After the pruning we have two types of variants, i.e., 1 AP and 2 AP, where the numbers denote how many activation layers have been pruned. Then, we compare the performance of these optimized architectures on the model inference accuracy and HE inference latency. The results are shown in Table 3.

Despite the architectures have different width (channels numbers), the original unoptimized architectures (w/o AP) have the best accuracy but also the highest latency. To deal with multiplicative depth of the unoptimized architectures, we have to set the polynomial degree in encryption parameter to 2^{15}, resulting in a tremendous latency increase. By applying AP, the 1 AP (one activation layer pruned) architecture only has a small accuracy loss (1.1-1.5%), while the latency has been greatly

Table 3: Ablation study for AP (Model Architecture Tradeoff) (AMA format)

Model		64-STGCN-3	128-STGCN-3
w/o AP	ACC	74.25%	75.31%
	Latency	4273.89s	10580.41s
w/ AP (1 AP)	ACC	73.12%	73.78%
	Latency	**1863.95s**	**4850.93s**
w/ AP (2 AP)	ACC	70.21%	71.36%
	Latency	1856.36s	4831.93s

Table 4: Ablation study of AMA format with batchsize=1

Model		64-STGCN-3	128-STGCN-3
Row-major format	Latency	2962.46s	9589.59s
	# of Ct	128	256
AMA	Latency	**1863.96s**	**4850.93s**
	# of Ct	100	200
Speedup		1.59×	1.97×

Table 5: Breakdown for HE operations in different layer of 64-STGCN-3

Layer	Row-major format				AMA			
	Rot	PMult	CMult	Add	Rot	PMult	CMult	Add
Spatial Conv	6.9K	623K	-	622K	6K	56K	-	55K
Temporal Conv	4K	295K	-	294K	7.7K	230K	-	230K
GlobalAvgPooling	832	-	-	896	28	-	-	96
FC	60	3.8K	-	240	240	240	-	240
Activation	-	1.3K	640	640	-	1K	500	1K

improved by about 2.3× due to two levels has been saved. This is because this two levels allow us to tradeoff the polynomial degree with a security level decrease and achieve a sweet point. We can use 2^{14} as our polynomial degree for the optimized depths with a 80-bit (previous 128-bit) security level. If we try to prune one more activation layer (2 AP), the accuracy loss (3.95-4.04%) for is larger than the 1 AP variant. Besides, the latency speedup is limited because we can not reduce the polynomial degree further to have at least 80-bit security level.

5.2 AMA Format Effectiveness

To evaluate the effectiveness of the AMA format method, we compare its performance against the row-major formatting with the same encryption parameter setting.

As the AP improves the latency for every HE operation, we evaluate the performance of two data formatting methods on AP optimized architecture. Then, we compare our AMA formatting with row-major format in different batchsize settings.

5.2.1 Compare with row-major format

As described in Table 4, our AMA format improves the inference latency by 1.59× for 64-STGCN-3 architectures, and 1.97× for 128-STGCN-3 architecture. Table 5 is a breakdown of HE operations for 64-STGCN-3. From the table we see that the AMA format has only 31.3% of PMult and Add, compared with row-major format. Here are two reasons for the reduction of operations.

First, AMA format uses fewer ciphertext than row-major format. The row-major format does not fully utilize the ciphertext space as the featuremap has a size of 256×25. In row-major format, the featuremap is first converted into a 1D vector with a size of 6400; then zero padding is applied to the right end of this vector to make the size equal to 8192. Therefore, in the resulted encrypted ciphertext, 1792 of slots has been wasted. For our AMA format, it fully utilizes the ciphertext space, because 256 is a power-of-two number. None of slots in ciphertext is wasted.

Second, compared to row-major formatting, our AMA formatting ciphertext could perform matrix-matrix multiplication with less HE operations. For the matrix used in our proposed network, one row-major format ciphertext should perform 19× multiplications for one output channel. However, one AMA formatting ciphertext only need to perform 3× multiplications for one output channel. Similar reason holds for the Add operation.

5.2.2 Different batchsize settings

We analyze our data formatting method in different batchsize settings. As described in Table 6, with the batchsize increasing, the latency of row-major format increases linearly as they do not have any

Table 6: AMA performance with batchsize increase

Model	Batchsize	Row-major	AMA	Average Latency	Speedup
64-STGCN-3	1	2965.46 sec	1863.95 sec	1863.95 sec	1.59 ×
64-STGCN-3	2	5931.92 sec	2704.41 sec	1352.21 sec	2.19 ×
64-STGCN-3	4	11852.84 sec	4390.66 sec	1097.66 sec	2.70 ×
64-STGCN-3	8	23703.68 sec	7770.91 sec	971.36 sec	3.05 ×
64-STGCN-3	16	45179.33 sec	14535.23 sec	**908.45 sec**	**3.10** ×

Table 7: Compare with the previous benchmarks on 64-STGCN-3

Method	Batchsize	HOC				
		Rot	CMult	PMult	Add	Total
CHET	1	16K	1.28K	1.3M	1.29M	2.61M
Fast-HEAR		12K	1K	923K	922K	1.86M
CryptoGCN		14K	500	287K	287K	589K
CHET	2	32K	2.56K	2.6M	2.59M	5.23M
Fast-HEAR		24K	2K	1.84M	1.84M	3.7M
CryptoGCN		**16K**	**1K**	**575K**	**574K**	**1.17M**

parallelism for a mini-batch setting. However, our AMA format allows processing a mini-batch of data on the same ciphertext, which allows the parallelism for a mini-batch setting. Besides, the number of rotations for multi-channel convolution decreases when the batchsize increases, the number of other HE operations is linearly increasing, resulting in a higher speedup up to 3.1× for average latency with the batchsize increasing.

5.3 Computation Complexity Evaluation

Table 7 compares CryptoGCN against the state-of-the-art privacy-preserving neural network frameworks (i.e., CHET [3] and Fast-HEAR [8]). The previous frameworks and ours were implemented in different environments (different CPUs and number of threads). For a fair comparison, we evaluate the numbers of the required homomorphic operations for 64-STGCN-3 on the same dataset, which are independent of the hardware and software configurations.

Similar to our work, CHET and Fast-HEAR utilize the ciphertext packing technique to reduce HE computation complexity. They used row-major format as the data representation for the feature maps. The main difference between Fast-HEAR and CHET is that Fast-HEAR leverage the non-valid space in ciphertext after down-sampling (avg pooling layer) in NN model such that Fast-HEAR could have less Homomorphic operation count (HOC).

Neither CHET nor Fast-HEAR is optimized for GCN-based models and the unique matrix multiplication mechanism could significantly increase the HOC. When batchsize equals to 1, compared to CHET and Fast-HEAR, our AMA encoded ciphertext better utilizes the sparsity of the matrix and significantly reduces the multiply and addition operations. Specifically, when performing matrix multiplication combining with a multi-channel convolution, AMA format avoids performing an inner-loop for matrix multiplication and hence reduces the amount of PMult and Add operations by 52.5-66.2%. Furthermore, we pack the graph data into ciphertexts to maximize the use of the slots and prune one activation layer from the original architecture, which reduces the number of CMult by 40-50%. With the batchsize increasing, CryptoGCN could reduce 77.4% of total HOC than prior work.

6 Conclusion

Homomorphic encryption has become an effective way to build the private-preserving machine learning thanks to the great development in HE scheme research. In this paper, we build a fast and scalable LHE-based private preserving inference framework optimized for GCN models by our novel AMA data formatting and model architecture optimization strategy. To the best of our knowledge, this is the first framework supporting private inference for a large skeleton joint data with a 40× size of previous work [8] and a 3× deep neural network model. Our solution shows highly promising

results for enhancing privacy-preserving inference on GCN models in a cost-effective manner. In the future, we could leverage multi-threading technique to further reduce the latency. We would like to extend the encrypted data to both graph node features and adjacency matrices, as these matrices may also contain a part of sensitive information. By encrypting graph node features and adjacency matrices, the clients' privacy could be fully guaranteed.

References

[1] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821. PMLR, 2019.

[2] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[3] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 142–156, 2019.

[4] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.

[5] Shai Halevi and Victor Shoup. Algorithms in helib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.

[6] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. Deepreduce: Relu reduction for fast private inference. In *International Conference on Machine Learning*, pages 4839–4849. PMLR, 2021.

[7] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[8] Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. Hear: Human action recognition via neural networks on homomorphically encrypted data. *arXiv preprint arXiv:2104.09164*, 2021.

[9] Qian Lou and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. Safenet: A secure, accurate and fast neural network inference. In *International Conference on Learning Representations*, 2020.

[11] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522, 2020.

[12] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[13] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th annual design automation conference*, pages 1–6, 2018.

[14] Microsoft SEAL (release 3.7). `https://github.com/Microsoft/SEAL`, September 2021. Microsoft Research, Redmond, WA.

[15] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1010–1019, 2016.

[16] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We clearly present the theoretical analysis and compare our contribution by ablation study in the result part
   (b) Did you describe the limitations of your work? [Yes] See section 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See section 3.1
   (b) Did you include complete proofs of all theoretical results? [Yes] See section A.1
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See in section 4
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See in section 4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We report the average latency with same random seed setting
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See in section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Yes, we cite all the creators in the papar
   (b) Did you mention the license of the assets? [Yes] SEAL has MIT license. ST-GCN repo has BSD-2-Clause license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Complete Explaination for Theoretical Complexity Compare

We continue to explain the theoretical results presented in Table 1. The assumption has been illustrated in 3.1

For row-major format:

For one ciphertext, it represent the data from one channel. In order to multiply with A dense matrix $J \times J$. We have to rotate each ciphertext with $2J - 2$ times.

Total $Rotation$= $B \times C \times (2J - 2)$

As we need to get C output channel data, all the existing BC ciphertext need to do $2J - 2$ times multiplications.

Total $PMult = C \times B \times C \times (2J - 2)$

Then, we need to get same BC ciphertext as the output channel is also C. We just sum up all the ciphertext.

Total $Add = C \times B \times C \times (2J - 2) - B \times C$

For AMA format:

The rotation operation is only used to sum up all the input channel data, because the ciphertext with AMA format contains J/B channels data. For each ciphertext, it need to rotate (J/B-1) times.

Total $Rotation$ = $B \times C \times (J/B - 1)$

To get C output channels data, each ciphertext need to perform J times $PMult$

Total $PMult$ = $B \times C \times J \times C$

Then, we need to get same $B \times C$ ciphertext as the output channel is also C. We just sum up all the ciphertext.

Total $PMult$ = $B \times C \times J \times C - B \times C$