SpENCNN: Orchestrating Encoding and Sparsity for Fast Homomorphically Encrypted Neural Network Inference

Ran Ran [1] **Xinwei Luo** [1] **Wei Wang** [2] **Tao Liu** [3] **Gang Quan** [4] **Xiaolin Xu** [5] **Caiwen Ding** [6] **Wujie Wen** [1]

Abstract

Homomorphic Encryption (HE) is a promising technology to protect clients' data privacy for Machine Learning as a Service (MLaaS) on public clouds. However, HE operations can be orders of magnitude slower than their counterparts for plaintexts and thus result in prohibitively high inference latency, seriously hindering the practicality of HE. In this paper, we propose a HE-based fast neural network (NN) inference framework–SpENCNN built upon the co-design of HE operation-aware model sparsity and the single-instruction-multiple-data (SIMD)-friendly data packing, to improve NN inference latency. In particular, we first develop an encryption-aware HE-group convolution technique that can partition channels among different groups based on the data size and ciphertext size, and then encode them into the same ciphertext by novel group-interleaved encoding, so as to dramatically reduce the number of bottlenecked operations in HE convolution. We further tailor a HE-friendly sub-block weight pruning to reduce the costly HE-based convolution operation. Our experiments show that SpENCNN can achieve overall speedups of 8.37×, 12.11×, 19.26×, and 1.87× for LeNet, VGG-5, HEFNet, and ResNet-20 respectively, with negligible accuracy loss. Our code is publicly available at https://github.com/ranran0523/SPECNN.

[1]Dept. of Electrical and Computer Engineering, Lehigh University [2]Anonym, Inc. [3]Dept. of Mathematics and Computer Science, Lawrence Technological University [4]Dept. of Electrical and Computer Engineering, Florida International University [5]Dept. of Electrical and Computer Engineering, Northeastern University [6]Dept. of Computer Science and Engineering, University of Connecticut. Correspondence to: Ran Ran <rar418@lehigh.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

1. Introduction

For the past decade, we have witnessed the tremendous progress of machine learning and the great success achieved in practical applications. Convolution Neural Network (CNN) (Alex et al., 2014) models, for example, have been widely used for many cognitive tasks such as medical imaging, face and human action recognition. Meanwhile, there is a growing interest in deploying machine learning on cloud as a service (MLaaS) (Varghese & Buyya, 2018; Marston et al., 2011). While cloud computing has been well recognized as an attractive solution, especially for computation-intensive scenarios such as the MLaaS, outsourcing sensitive data and data processing to the cloud service provider can pose a severe threat to the client's privacy.

Homomorphic Encryption (HE) (Rivest et al., 1978) is a promising technology for protecting clients' privacy when deploying MLaaS on the cloud. HE allows computations to be performed on encrypted inputs. The decrypted output matches the corresponding results computed from the original inputs. Thus, a client can encrypt the sensitive data locally and send the encrypted ciphertexts to the cloud. The encrypted results sent from the cloud can be correctly decrypted using the secret key held by the client. While HE helps maintain the confidentiality of the computation process on the cloud effectively, one major problem is the excessive computational cost associated with the HE operations over the encrypted data (e.g. HE multiplication, addition, rotation). In fact, HE operations can be several (i.e., three to seven) orders of magnitude slower than the corresponding operations on plaintexts (Jung et al., 2021), becoming the bottleneck that hinders the popularity of MLaaS.

One of the most effective approaches (e.g. Gilad-Bachrach et al. (2016); Brutzkus et al. (2019); Dathathri et al. (2019); Kim et al. (2022); Lee et al. (2022a)) to reduce the HE computational cost is to take advantage of the single-instruction-multiple-data (SIMD) capability, supported by HE schemes, e.g. Cheon-Kim-Kim-Song (CKKS) (Cheon et al., 2017) and Brakerski/Fan-Vercauteren (BFV). Smart & Vercauteren (2010) packs multiple data elements into different "slots" in the same ciphertext and thus computations for data elements at the same slot of two encoded messages can be performed in parallel. The challenge is how to pack data based on the

characteristics of the applications so that computation can be conducted effectively in a SIMD manner. In particular, the problem rises when the computation needs to be performed on data elements at different slots of the messages. To re-arrange the location of each individual data element in an encrypted message is out of the question due to its large overhead. A more reasonable solution is to employ the HE-rotation[1] operation that can move the data element cyclically in the same message. However, HE-rotation has a high latency cost due to the required permutation and key-switching operation, compared with other HE-operations such as the HE multiplication of a ciphertext with a plaintext (HE-PMult) and HE addition of two ciphertexts (HE-Add), as shown in Figure 1(a). Therefore, how to judiciously encode the inputs and perform the SIMD operations plays a key role in reducing the HE computation complexity.

In this paper, we study the problem of how to improve the HE-based inference latency when deploying a privacy-preserving machine learning (PPML) platform based on CNNs on the cloud. In particular, **this work focuses on reducing HE operations involved in convolution layers** based on the fact that such layers often dominate the computation workload in CNN inference. While there exist other studies in this regard, e.g. optimizing the costly bootstrapping operation (Lee et al., 2022a;b), this work is orthogonal to these studies and represents another important aspect for accelerating the HE-based inference.[2]

Assuming the client inputs (e.g., images) are encrypted as the ciphertexts and associated CNN model parameters are encoded as plaintext messages, the major HE computations in CNN inference are therefore HE-PMult, HE-Add, and HE-rotation operations. Traditional neural network optimization techniques such as sparsification and pruning (Han et al., 2015b; Wen et al., 2016; Frankle & Carbin; Liu et al.; Li et al., 2019) help to reduce the computation and memory overhead to speedup CNN inference in the plaintext domain. However, this does not hold in the HE-based inference. In fact, they achieve very limited or even no improvement compared to dense models in their encrypted counterpart, as we shall show in Section 2.3. **Because simply extending the design principles of existing sparsity solutions without considering the special SIMD-friendly data packing requirements for the encrypted inference, can reduce neither the huge volume of expensive HE operations nor the much-increased memory footprint for computing activation ciphertexts in each layer.** As an example, they are unable to principally reduce the more expensive HE rotation involved frequently in the multi-channel convolution computation. Note that the computation overhead of an HE rotation can be over 43× of that for a HE-Pmult or a HE-Add operation, as shown in Figure 1(a),

To this end, we develop a HE-based CNN inference framework, i.e., SpENCNN, with the goal of effectively exploiting the SIMD feature of the HE scheme to improve the CNN inference latency. In particular, we develop two techniques to reduce the HE computational cost. **First**, we develop HE-group convolution and associated group-interleaved encoding to optimize channel locations on ciphertexts based on the number of convolutional groups and ciphertext size, thus significantly reducing the number of costly HE-rotations. **Second**, we further optimize the model architecture by pruning and training the weights in the sub-blocks iteratively with the goal to minimize HE-rotations and accuracy loss. To comprehensively evaluate the efficiency of SpENCNN, we have conducted experiments on the following two settings using the state-of-the-art leveled HE–CKKS and the MNIST and CIFAR-10 datasets: *1) CKKS-based PPML for CNN models with no need of bootstrapping; 2) CKKS based PPML for deeper models with the support of the-state-of-the-art bootstrapping solution (Lee et al., 2022a).* For the former, SpENCNN can achieve overall speedups of 8.37×, 12.11× and 19.26×, for LeNet, VGG-5 and HEFNet, respectively, with negligible accuracy loss. For the latter, while bootstrapping contributes significantly to the overall inference latency in ResNet-20, SpENCNN still offers 1.87× speedup while preserving the accuracy by reducing the HE rotations. *To our best knowledge, this is the first work that builds an optimizing framework for CNN model architecture from the aspect of co-design of structural sparsity and data packing in HE to benefit HE-based PPML inference.*

2. Preliminary

2.1. CKKS Homomorphic Encryption

Homomorphic Encryption (HE) allows computations to be performed on encrypted data without decryption. Among various HE schemes, the CKKS (Cheon et al., 2017) is widely adopted in the encrypted neural network inference because of supporting the fixed-point real number arithmetic and potentially avoiding the prohibitively expensive bootstrapping. The CKKS-based HE operations mainly consist of ciphertext addition HE-Add $(ct_1 + ct_2)$, ciphertext multiplication HE-Cmult $(ct_1 \times ct_2)$, scalar multiplication HE-Pmult $(pt_1 \times ct_2)$, ciphertext roation HE-rotation $Rot(ct, k)$, etc. For MLaaS that only encrypts clients' data, HE-Add, HE-Pmult and HE-rotation often dominate the computations of an encrypted inference. Among these three operations, HE-rotation costs much longer latency than the other two, e.g. $\sim 43\times$ as our profiling result in Figure 1 (a) shows, due to the complex automorphism operation and a key-switching operation. The detailed calculation process of HE-rotation

[1] E.g., $Rot(ct, k)$ transforms an encryption of $(v_0, ..., v_{N/2-1})$ into an encryption of $(v_k, ..., v_{N/2-1}, v_0, ..., v_{k-1})$.

[2] We show the efficiency of our work to speedup PPML in the presence of costly bootstrapping in Section 4.

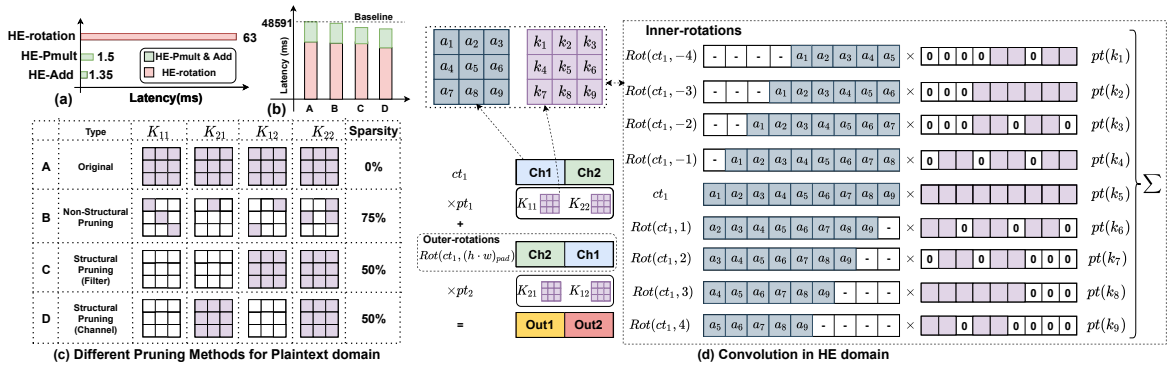

Figure 1. (a) Comparison of different HE-operations' latency. (b) Comparison of the HE convolution latency under different pruning methods. (c) Illustration for different pruning methods for plaintext domain. (d) Multi-channel convolution process in HE domain. Notation definitions refer to section 3.1. $pt(k_i)$ indicate the weight plaintext. The convolution layer used here has 64 input- and 64 output-channel, with a 3×3 kernel. The input feature map size of the convolution layer is 32×32.

can be described as:

$$Rot(ct, k) = (c(X^{ik}), 0) + P^{-1}(a(X^{ik}) \cdot evk^k_{rot}) \quad (1)$$

where the evaluation key (evk^k_{rot}) is a public key with a larger modulus PQ, and P is greater than Q. Assume $ct = (c(X^i), a(X^i))$ represents a ciphertext before rotation, then the automorphism ($c(X^{ik})$ and $a(X^{ik})$) maps each polynomial coefficient index i to output polynomial coefficient index $ik \ mod \ N$, where N is the polynomial degree. The second term on the right side of Equation 1 represents the key-switching operation to ensure the final ciphertext can be still decrypted by the same secret key. It is very expensive and could take over 90% of all operations in practice (Samardzic et al., 2022).

2.2. Threat Model

We assume the cloud-based machine learning service, of which a trained convolutional neural network (CNN) model with plaintext weights, is hosted in a cloud server. A client could upload one's private and sensitive data to the public cloud for obtaining an online inference result. The cloud server is semi-honest (e.g., honest but curious). To ensure the confidentiality of clients' data against such a cloud server, the client utilizes HE to encrypt the data and then send it to the cloud for performing encrypted inference without decrypting the data or accessing the private key. Finally, the client can decrypt the returned encrypted inference results from the cloud using a private key. In this work, we focus on encrypting the client's data, and others like model parameters, are assumed as plaintext.

2.3. Motivation Example

To identify the computation bottleneck in HE inference, we analyze the computation pattern of the convolutional layer, which often dominates CNN inference's memory and computational overheads, in the encryption process. Here the input and output activation feature maps are encrypted as ciphertext, while the convolutional kernels are assumed as plaintext. We also assume the state-of-the-art ciphertext encoding–row-major (Dathathri et al., 2019; Kim et al., 2022) is adopted here. This allows efficient multi-channel ciphertext packing to take advantage of the CPU's SIMD architecture for fast HE inference. Figure 1 (d) shows the typical HE convolution process of a convolution layer which consists of 2-input/output channels with 3×3 kernels. To compute a ciphertext output feature map, two types of ciphertext rotations need to be performed sequentially. First, **inner-rotation** rotates each input channel's ciphertext feature map 8 times (or $K^2 - 1$, here kernel size $K = 3$). Each rotated version will need to be multiplied with its corresponding weight plaintext, and then such results will be summed up to obtain an intermediate ciphertext from each input channel, which will further be concatenated as a whole ciphertext (e.g. Ch1 and Ch2 as **ct1**). Second, **outer-rotation** rotates the concatenated ciphertext multiple times (in this simple example, 1 time because of packing 2 output channels as a ciphertext). Finally, all ciphertext output feature maps can be obtained in parallel by the summation of these rotated copies. Apparently, compared to non-encrypted convolution, HE convolution significantly escalates the memory and computation overheads. Moreover, since the latency of HE-rotation can be much higher than other operations due to complex automorphism and key switching operations (see our profiling result in Figure 1(a) 63ms for HE-Rot v.s. 1.5ms for HE-Pmult, detailed setting in Sec. 4.1), and the multi-channel convolutions in deep CNNs would involve a huge volume of HE-rotation [3]. As a result, the long-latency HE-rotation quickly becomes a bottleneck of the encrypted inference.

One straightforward solution to accelerating HE inference

[3]The matrix-vector multiplications in fully-connected layers also require a substantial amount of HE-rotation.

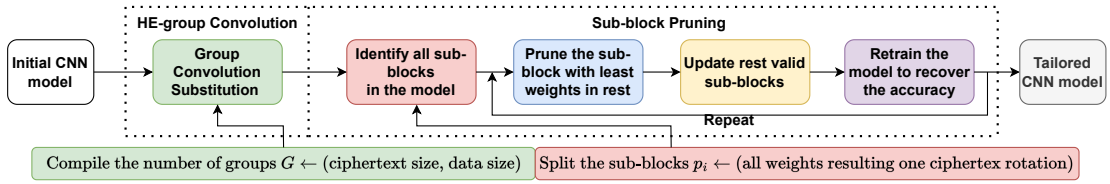

Figure 2. An overview of SpENCNN for optimizing HE-based CNN inference, mainly consists of two orthogonal techniques to generate a tailored model: (1) HE-group Convolution (outer-rotation optimization), and (2) Sub-block Pruning (inner-rotation optimization).

is to reduce the number of rotations by zeroing out (or pruning) the plaintext weights. As Figure 1(c) shows, if any weight plaintext–$pt(k_i)$ contains all zero values, then the corresponding ciphertext rotation–$Rot(ct_1, k)$ and its associated multiplication and summation can be safely eliminated. Since existing pruning techniques have been proven to be effective in reducing the computation and memory overhead to speedup the non-encrypted inference without accuracy drop, we apply two representative pruning methods–non-structured pruning (Han et al., 2015a) (zeros appear randomly in a kernel, see Figure 1 (c)–B, 75% sparsity) and structured pruning (Wen et al., 2016) (structured zeros in a kernel, see filter pruning and channel pruning in Figure 1 (c)–C and D, 50% sparsity). For HE operations in an example convolutional layer with 64 input/output channels, feature map size 32×32 and kernel size 3×3, as Figure 1 (d) shows, the existing pruning achieves very marginal or even no reduction of the HE-rotation latency which dominates the convolution computation. In the worst case, it even cannot remove any HE-rotation despite the high model sparsity, e.g. the non-structured pruning with 75% sparsity ratio. *The underlying reason is two-fold: 1) pruning is unable to address the outer-rotation since computing an output ciphertext by convolution needs to sum all channels' feature maps belonging to the same ciphertext if using the state-of-the-art ciphertext encoding (see Figure 3 (a)); 2) existing pruning techniques are designed for non-encrypted inference, and the special channel-wise ciphertext operations involved in HE convolution are ignored.* This prompts the need to jointly optimize ciphertext encoding and encryption-aware model sparsity to accelerate HE inference.

3. The SpENCNN Framework

In this section, we present the technical details of the SpENCNN framework. Figure 2 depicts an overview of the SpENCNN framework, of which its input is an initial CNN model and the output is a tailored CNN model suitable for fast HE inference. To achieve this, it requires two processing stages: **(1) HE-group Convolution**, which is designed to reduce the outer-rotations caused by multi-channel encoded ciphertext (Kim et al., 2022; Lee et al., 2022a). In particular, we design an adjustable method and determine a theoretically optimal group number G_{base} based on the size of the ciphertext and data packed in the CKKS HE scheme, to ensure that all outer-rotations can be eliminated while keeping model accuracy. **(2) Sub-block Pruning**, which is further proposed to reduce the number of inner-rotations. However, this is non-trivial. We observe that to reduce as many inner-rotations as possible, we must precisely identify and completely prune selected sub-blocks. This, unfortunately, results in a considerable accuracy drop. To address the challenge, we develop a set of sub-steps which include identifying sub-blocks, pruning sub-blocks, updating the remaining sub-blocks, and retraining for accuracy recovery.

3.1. HE-Group Convolution

Two intuitions. Our proposed HE-group convolution is based on two intuitions. We observe that the number of required outer-rotations for a ciphertext is $Rot_{outer} = {}^{N}/_{2\times(h\times w)_{pad}} - 1$, where N is the polynomial degree defined in the cryptographic parameters. h and w are the height and width of the input feature map. pad rounds a number to the next power of two. Each ciphertext generated by the outer-rotation further requires a set of inner-rotations. The number of inner-rotations is $Rot_{inner} = K^2 - 1$, where k is the convolutional kernel size. Apparently, increasing the number of outer-rotation–Rot_{outer} by just 1 can bring an extra $Rot_{inner} = K^2 - 1$ inner-rotations. This gives us the first intuition–*reducing the number of outer-rotations will fundamentally reduce the computational overhead.*

The reason behind the outer-rotation is that multiple channels on the same ciphertext are involved in the same multi-channel convolution. In our study, we find that convolution (e.g., depthwise convolution (Howard et al., 2017)) can be also performed individually within each single channel. We also find that the group convolution technique (Krizhevsky et al., 2012; Zhang et al., 2018; Ioannou et al., 2017) can reduce the number of channels involved in each group. This gives us the second intuition–*reducing the number of channels in the same group can eliminate the outer-rotation.*

Based on these intuitions, we propose the HE-group convolution. Given the group number G, the upper bound of the number of channels in the same group can be $\lceil ({}^{N}/_{2\cdot(h\cdot w)_{pad}}) \times {}^{1}/_{G} \rceil$. Then the relationship between Rot_{outer} and G can be further expressed as $Rot_{outer} = \lceil ({}^{N}/_{2\cdot(h\cdot w)_{pad}}) \times {}^{1}/_{G} \rceil - 1$. Accordingly, we can go through the power of 2 numbers of G from 1 to the optimal value $G_{base} = {}^{N}/_{2\cdot(h\cdot w)_{pad}}$ to cancel the Rot_{outer} with negligible

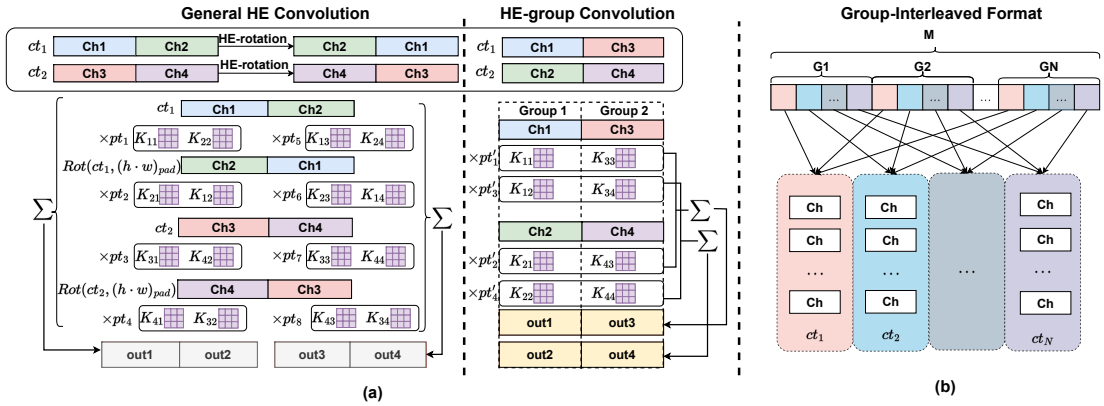

Figure 3. (a) an example of HE convolution which generates 4 output channels from the 4 input channels with 3×3 kernels. By HE-group convolution, we only need 2 group-interleaved encoded cts to multiply with corresponding pts and sum up to get the output channels data. (b) proposed HE-group convolution by a group-interleaved format.

model accuracy loss in our design. Theoretically, this optimized value indicates zero outer-rotations when G reaches the upper bound.

Group-interleaved encoding. However, we find that the traditional ciphertext encoding format is not compatible when we implement our grouping idea. This is because the row-major format is mainly designed to perform convolution in the SIMD manner without considering the channel positions on the ciphertext. To address this issue, we propose a **group-interleaved encoding** format–the channel data from different groups are placed on the same ciphertext in an interleaved manner. This new encoding facilitates fast HE-group convolution without involving any outer-rotation.

Figure 3 shows an example of proposed HE-group convolution and group-interleaved encoding. We assume that two ciphertexts contain 4 channels of data, and each ciphertext ct_i contains 2 channels. As shown in Figure 3 (a)–left, in general HE convolution, each ct_i has to perform 1 outer-rotation to cover the 2 different channels, i.e., $\{ch1, ch2\}$ and $\{ch2, ch1\}$ for ct_1. Convolution will be performed individually on each outer-rotated case for all ciphertexts. The encrypted output channels $\{out_1, out_2\}$, $\{out_3, out_4\}$ can be generated after a summation.

For our HE group convolution, as Figure 3 (a)–right shows, sibling channels $\{ch1, ch2\}$ and $\{ch3, ch4\}$ from the same ct_i are in different convolution groups, which are encoded by our group-interleaved format, i.e., $\{ch1, ch3\}$ in ct_1 and $\{ch2, ch4\}$ in ct_2. Now, the outer rotation is eliminated because each ct_i can perform 2 groups of convolution individually without rotating the channels. The encrypted output channels after summation, i.e., $\{out_1, out_3\}$, $\{out_2, out_4\}$, are naturally group-interleaved and can be immediately sent to next HE-group convolution. Figure 3 (b) further shows the generalized group-interleaved encoding, in which a ciphertext can encrypt M channels using the adjustable group number G, with constraints $G \leq M$ and $M|G = 0$.

3.2. Sub-block Pruning

We design the sub-block pruning to further remove the remaining inner-rotations after the HE-group convolution. Our idea is to prune (zero out) a whole set of weights corresponding to specific inner-rotations, so that the computational overhead of these inner-rotations can be further reduced. In HE-convolution, an inner-rotated ciphertext will be multiplied with the weights at the same position, namely "sub-block", from all relevant kernels. Therefore, our design tends to cut out the same sparse pattern on these sub-blocks for all relevant kernels of the same ciphertext. As the example in Figure 4 (a) shows, the 4 kernels of ct_1 share the same sparse pattern, thus eliminating 6 inner-rotations.

To obtain the desired sparse pattern, we propose sub-block pruning. The more sub-blocks p_i being pruned, the fewer inner-rotations needed. On the other hand, pruning more sub-blocks will lead to more accuracy drop. It is an optimization problem, in which, we need to minimize the total number of sub-blocks while maintaining the prediction accuracy concurrently:

$$\begin{aligned} &\min\{P = \sum_i p_i \cdot I_i\} \\ &\quad \text{s.t. } Acc(f(x;(W,P))) \geq Acc(f(x;W)) \\ &\quad \text{where } I_i = \begin{cases} 0 & p_i \textit{ is pruned} \\ 1 & other \end{cases} \end{aligned} \tag{2}$$

We aim to find the sub-blocks with the minimum weight importance to the model at each iteration and prune, then, retrain the model for a few epochs to recover the accuracy. However, the sizes of sub-blocks are different. To measure the weight importance of each sub-block in a fair way, we define the weight importance metric as an average L^2 norm $\frac{||w_{p_i}||}{dim(w_{p_i})}$. As described in the following Algorithm 1, at each iteration, we prune the sub-blocks with the least weight importance. The iterative algorithm would stop when the

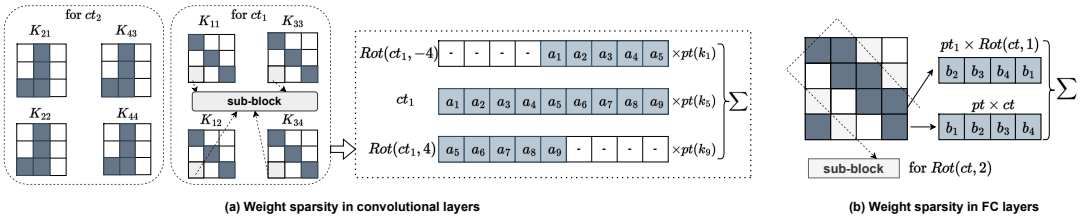

Figure 4. (a) The weight sparse pattern in convolutional layers. For the same ct, their weight sparse patterns must be the same. For different ct, the weight sparse pattern may change. (b) The weight sparse pattern in FC layers is in a diagonal-wise shape.

model accuracy is lower than the initial accuracy.

The proposed sub-block pruning can be also extended to the FC layer. Given a N-element $ct = \{x_{1..N}\}$, the weight matrix of an FC layer can be reshaped as a $M \times N \times N$ tensor $\{\vec{W}\}$ (zero padding if needed). To multiply an $N \times N$ matrix, we can multiply the diagonal N elements (as plaintext) with each copy of ct for total N times (including $N-1$ rotated copies and an original ct) according to the diagonal-wise matrix multiplication with ct (Halevi & Shoup, 2014). To illustrate this, we provide a toy example in Figure 4 (b). Assuming we have $ct = \{b_{1..4}\}$ and a 4×4 weight matrix. Without sub-block pruning, the ciphertext ct needs to rotate 3 times to perform the weight matrix (plaintext)-ciphertext multiplication. However, by applying sub-block pruning and removing two diagonal sub-blocks (indicated by the white blocks in the figure), we can save 2 rotations. The final computation can be expressed as $pt \times ct + pt_1 \times Rot(ct, 1)$, where pt represents the central diagonal weights, and pt_1 represents the remaining weights.

Algorithm 1 Sub-blocks Iterative Pruning

Input: CNN model: $f(x; (W, P))$,
Remark: x-Data, W-Weights, P-HE blocks
Output: HE-friendly model: $f(x; (W', P'))$
$P' = P = \sum_i p_i \cdot I_i$
While Accuracy loss ≤ 0 :
  $i \leftarrow \underset{i}{\operatorname{argmin}} \frac{||w_{p_i}||}{dim(w_{p_i})}$
  $I_i = 0$
  prune weights in p_i **from current** P
  update P'
  retrain model with P' and update W'
end While
Return $f(x; (W', P'))$

4. Evaluation

4.1. Experiment Setup

Setup. We conduct our experiments on a workstation equipped with an AMD Ryzen Threadripper 3975WX CPU, an NVIDIA RTX 3090 GPU, and 256GB of RAM. To evaluate our proposed SpENCNN, we select four baseline CNN models that are typically adopted in HE inference performance evaluation and implement them using PyTorch on GPU. This includes the LeNet-like for MNIST (2016), the VGG-5 (Rathi et al., 2020), the HE-friendly Net (HEFNet), and the ResNet-20 (Idelbayev) for CIFAR10. The first three models and the fourth model are used for evaluating the cases without and with bootstrapping, respectively. The following Table 2 contains the detailed convolutional kernel size, weight matrix size, and number of channels. These three baseline models have different properties. The LeNet-like model is a tiny model designed for simple classification tasks so that the channel number and weight matrix size are small and have the least number of layers. The VGG-5 model has much more weight in FC layers and the max number of layers. The HEFNet contains the most convolutional layers and the widest channel size. The ResNet-20 architecture is consistent with the published version (He et al., 2016).

We adopt the Stochastic Gradient Descent (SGD) optimizer with a mini-batch size of 64 on MNIST and 128 on CIFAR10, a momentum of 0.9, and weight decay of e^{-4} to train each selected model for 100 epochs. The initial learning rate is set to 0.01 with a decay factor of $5e^{-4}$. For average pooling or convolution with stride > 1, we use the same implementation as CHET (Dathathri et al., 2019) for the first three models and (Lee et al., 2022a) for the ResNet-20. Since the non-linear activation function like ReLU cannot be evaluated in HE, we replace ReLU with the adaptive quadratic polynomial function $f(x) = ax^2 + bx + c$ following the recent work (Peng et al., 2022), where a, b, c are trainable parameters to maintain the model accuracy. Table 1 lists the specifications of these models and the corresponding accuracy. In particular, the test accuracies for LeNet-like-MINST, VGG-5-CIFAR10, HEFNet-CIFAR10,

Table 1. The four baseline CNN models and corresponding Encryption Parameters. LeNet-like is for the MNIST dataset. VGG-5, HEFNet and ResNet-20 are for CIFAR-10 dataset.

Model	# Layers			Groups	Accuracy	w/o Retraining
	Conv	FC	Act	(G_{base})	(%)	Accuracy (%)
LeNet-like	2	2	3	4	98.95	98.05
VGG-5	3	3	5	8	84.06	78.31
HEFNet	4	1	4	8	83.67	79.91
ResNet-20	19	1	19	8	91.52	85.23
Model	Encryption Parameters				Mult	Security
	N	Q	P	q	Level	Level
LeNet-like	8192	264	24	24	10	104 bits
VGG-5	16384	529	31	33	16	104 bits
HEFNet	16384	436	31	33	13	128 bits
ResNet-20	65536	1501	46	51	58	153 bits

Table 2. Convolutional layer and Fully-connected layer size in three baseline models.

Network	Layer	# Input channel	# Output channel	Kernel size (Matrix size in FC)
LeNet-like	Conv1	1	32	5×5
	Conv2	32	64	5×5
	FC1	64	32	64×32
	FC2	32	10	32×10
VGG-5	Conv1	3	64	3×3
	Conv2	64	128	3×3
	Conv3	128	128	3×3
	FC1	8192	4096	8192×4096
	FC2	4096	4096	4096×4096
	FC3	4096	10	4096×10
HEFNet	Conv1	3	64	3×3
	Conv2	64	128	3×3
	Conv3	128	256	3×3
	Conv4	256	256	3×3
	FC1	1024	10	1024×10
ResNet-20	Conv1	3	16	3×3
	Conv2 ×6	16	16	3×3
	Conv3	16	32	3×3
	Conv4 ×5	32	32	3×3
	Conv5	32	64	3×3
	Conv5 ×5	64	64	3×3
	FC1	64	10	64×10

ResNet-20-CIFAR10, are 98.95%, 84.06%, 83.67%, and 91.52%, respectively. Besides, we find that without retraining, the models would suffer from a considerable accuracy drop after pruning, while with retraining, the models can reach accuracy comparable to their published versions. Hence, we prune one block in each iteration and then perform 5 epochs of fine-tuning to recover model accuracy.

We use Microsoft SEAL library v3.4.5 (SEAL) to implement the RNS variant of CKKS (Cheon et al., 2019) based HE inference for these networks. Table 1 also lists the key parameters used in our RNS-CKKS encryption, including the polynomial degree N, the total modulus in bit-length Q, the scale factor in bit-length P and special modulus in bit-length q to maintain the HE evaluation accuracy, and total multiplication level. These parameters guarantee a security level of 104 bits for LeNet-like and VGG-5, 128 bits for HEFNet, and 153 bits for ResNet-20, which are sufficient to tolerate the known attack proposed by (Albrecht et al., 2015). For ResNet-20, the bootstrapping time is calibrated by the data provided by (Lee et al., 2022a) under a similar execution environment. As we replace the ReLU with degree-2 polynomials, the ResNet-20 model in this work has a lower level consumption than the original one in (Lee et al., 2022a), which reduces the latency of non-optimized ResNet-20 from 2275s to 647s.

Methodology. We first perform an ablation study to evaluate each individual technique's effectiveness and then compare the whole SpENCNN framework with the state-of-the-art methods. We adopt the average inference latency (in seconds) as the major metric. An image set containing 20 different samples is used to measure and report the latency for these models. A lower latency indicates better performance. In addition, we measure the left homomorphic operation count (HOC, in %), sparsity (in %), and accuracy (in %) of the tailored models. The lower HOC and lower sparsity while offering higher accuracy are desired on all models.

Table 3. Ablation study of HE-group convolution with the different number of convolution groups.

Model	Groups	HOC Left (%) Rot	HOC Left (%) Others	Accuracy (%)	Latency (s)	Speedup (×)
LeNet-like	1-baseline	-	-	98.95	1.2658	-
	2	51.52	52.91	98.95	0.6806	1.86
	4	**27.27**	**28.24**	**98.95**	**0.3807**	**3.32**
	8	27.27	16.47	98.67	0.3044	4.16
VGG-5	1-baseline	-	-	85.16	53.909	-
	4	87.53	84.08	84.53	46.539	1.16
	8	**85.45**	**81.42**	**84.06**	**45.311**	**1.19**
	16	85.45	80.10	82.23	45.053	1.20
HEFNet	1-baseline	-	-	84.91	24.113	-
	4	24.53	25.74	84.35	6.2491	3.86
	8	**11.95**	**13.36**	**83.67**	**3.2718**	**7.37**
	16	11.95	7.18	80.06	2.3627	10.21
ResNet-20	1-baseline	-	-	91.52	647	-
	2	51.4	52.72	91.43	475	1.36
	4	**27.11**	**28.76**	**90.21**	**392**	**1.65**
	8	14.96	15.12	85.31	351	1.84

4.2. Results

4.2.1. Evaluation on HE-Group Convolution

Table 3 lists our evaluation results for the HE-group convolution. We apply the HE group convolution alone (in ablation) to each baseline model and evaluate its effectiveness and scalability. In particular, we go through the number of groups from its default value (i.e., 1-baseline) until it exceeds its G_{base} according to our design in Section 3.1 (i.e., the highlighted 4, 8, and 8 for LeNet-like, VGG-5, and HEFNet, respectively). The G_{base} for ResNet-20 should be 16, but we could not set the group number to G_{base} due to prominent accuracy loss. Therefore, we select a moderate group number for ResNet-20 with desirable accuracy for comparison. For detailed analysis, we break down HOC into "Rot" (HE-rotation) and "Others" (other operations including HE-Pmult and HE-add).

Our HE-group convolution can be scaled to any convolutional model. As the number of groups increases, it can effectively reduce the number of HOC and maintain the accuracy, thus reducing the latency of HE inference and improving the performance. As listed in Table 3, the number of HE-rotation is reduced from 100% to 27.27%, 85.45%, 11.95%, and 27.11% on LeNet-like, VGG-5, HEFNet, and ResNet-20, respectively.

Once G_{base} is reached, the number of HE-rotation does not decrease further despite increasing the number of groups. This is because the outer-rotation is completely eliminated in HE-group convolution after the group number reaches G_{base}. We also find that HE group convolution can reduce other HOC such as HE-Pmult and HE-add even after exceeding G_{base}. This also contributes to latency improvement.

For example, the HE-group convolution is particularly effective on our HEFNet (i.e., $\sim 88\%$ and $\sim 86\%$ reduction for

Table 4. Ablation study of sub-block prune and comparison with other pruning methods.

Network	Groups	HOC Left (%) Rot	HOC Left (%) Others	Sparsity (%)	Latency (s)	Speedup (×)
LeNet-like	Dense-Baseline	-	-	0.00	1.2658	-
	NS-prune	96.12	96.23	91.00	1.2190	1.04
	S-prune (channel)	88.03	92.82	53.77	1.1202	1.13
	Sub-block prune	**35.21**	**34.07**	**63.83**	**0.4644**	**2.62**
VGG-5	Dense-Baseline	-	-	0.00	53.909	-
	NS-prune	97.59	97.14	91.88	52.5280	1.03
	S-prune (channel)	98.47	98.08	90.48	50.7178	1.06
	Sub-block prune	**15.89**	**16.11**	**89.87**	**8.7659**	**6.15**
HEFNet	Dense-Baseline	-	-	0.00	24.113	-
	NS-prune	85.60	88.97	72.95	21.1660	1.14
	S-prune (channel)	94.69	95.24	51.91	22.9240	1.05
	Sub-block prune	**41.88**	**38.11**	**63.90**	**9.3709**	**2.57**
ResNet-20	Dense-baseline	-	-	91.52	647	-
	NS-prune	90.23	91.82	78.21	599	1.08
	S-prune (channel)	96.21	96.84	53.12	628	1.03
	Sub-block prune	**52.31**	**50.12**	**56.40**	**475**	**1.36**

HE-rotation and others, respectively). This is because it has the largest volume of convolution layers among the selected models. Such a dramatic reduction in HOC further shortens the inference latency from 24.11s to 3.27s, which represents a 7.37× speedup. In contrast, the HE group convolution is the least effective in VGG-5, as it contains three large FC layers (size of 8192×4096), which cannot be substantially optimized using HE group convolution alone. There are more than 85% of HE-rotations and 80% of other operations that cannot be eliminated. And this number saturates as a lower bound after reaching the G_{base}, resulting in a limited speedup of 1.19×. For ResNet-20, the HE-group convolution can also effectively reduce a large number of HE operations because of the huge volume of convolutional layers. However, due to its long bootstrapping time of 275s, it only achieves a speedup of 1.65×.

We also observe that model accuracy slightly decreases as the number of groups increases. This is due to the fact that fewer channels are involved in the HE-group convolution compared to the general convolution (see Figure 3). Fortunately, our design is adjustable, allowing a trade-off between accuracy and the optimized number of convolution groups. For an optimized group number, the accuracy loss can be marginal (i.e., 0%, 1.1%, 1.2%, and 1.31% on LeNet-like, VGG-5, HEFNet, and ResNet-20, respectively).

4.2.2. Evaluation on Sub-block Pruning

We also evaluate the sub-block pruning alone (in ablation) and compare its effectiveness against the representative pruning methods that are widely used in non-encrypted inference, such as Non-structural prune (NS-prune) (Han et al., 2015a), and Structural-prune (S-prune) (Wen et al., 2016). Table 4 reports the results. Here we do not report the accuracy but the sparsity ratio and latency since all models maintain the original accuracy after applying pruning methods.

Our sub-block pruning can effectively improve the HE inference performance on all baseline models (i.e., the speedup

Table 5. Comparison with Hunter on model HOC left, sparsity, accuracy, latency, and speedup.

Network	Method	HOC Left (%) Rot	HOC Left (%) Others	Sparsity (%)	Accuracy (%)	Latency (s)	Speedup (×)
LeNet-like	Baseline	-	-	0	98.95	1.2658	-
	Hunter	40.95	39.91	59.99	98.95	0.5353	2.36
	Ours-4	**8.54**	**9.88**	**62.62**	**98.95**	**0.1535**	**8.37**
VGG-5	Baseline	-	-	0	85.16	53.909	-
	Hunter	17.86	18.93	89.81	84.03	9.9916	5.40
	Ours-8	**7.86**	**7.72**	**91.97**	**84.07**	**4.3830**	**12.11**
HEFNet	Baseline	-	-	0	84.91	24.113	-
	Hunter	48.27	42.20	57.82	83.63	10.855	2.22
	Ours-8	**3.99**	**4.61**	**65.62**	**83.67**	**1.2520**	**19.26**
ResNet-20	Baseline	-	-	0	91.52	647	-
	Hunter	51.12	52.39	48.12	90.20	461	1.40
	Ours-4	**14.10**	**15.47**	**53.32**	**90.21**	**344**	**1.87**

of 2.62×, 6.15×, 2.57×, and 1.36× on LeNet-like, VGG-5, HEFNet, and ResNet-20, respectively), which significantly outperforms other traditional pruning methods (i.e., marginal $\sim 1.1\times$ speedup on most cases). The reason is that our design is more HE-oriented and thus is effective in the ciphertext domain while traditional prunings are designed for achieving high sparsity ratio in the plaintext domain without considering the special requirement of ciphertext operations. For example, although NS-prune can prune $\sim 92\%$ of the weights on VGG-5, it cannot eliminate the HE overhead (i.e., $\sim 96\%$ HOC) caused by the remaining $\sim 8\%$ of the weights.

Our sub-block pruning method performs the best (i.e., $\sim 16\%$ HOC) on VGG-5 because it effectively eliminates the inner-rotations caused by the large number of redundant weights in the FC layers. Together with the previous results (see Table 3), sub-block pruning can be a good complement to the HE-group convolution that performs weakly on the FC layers. We also note that our sub-block pruning method on LeNet-like (i.e., 35.21% Rot left) slightly outperforms HEFNet (i.e., 41.88% Rot left). This is because the larger convolutional kernel (i.e., 5×5) in LeNet-like offer more space to optimize the inner-rotations using our method. For ResNet-20, the speedup is limited due to the bootstrapping overhead. However, the sub-block prune is still effective for reducing HE operations because of the increased number of convolutional layers.

4.2.3. Compare with SOTA solutions

We compare our method with the state-of-the-art HE-prune method–Hunter (Cai et al., 2022). The comparison results are presented in Table 5. In this evaluation, we combine the HE-group convolution and sub-block pruning and use the proposed G_{base} as the group number (i.e., highlighted data). For a fair comparison, pruning is controlled to ensure that our method and Hunter have the same level of accuracy (i.e., error$\leq \pm 0.04\%$) on all baseline models. We can see from Table 5, our method outperforms the state-of-the-art significantly in terms of HOC, sparsity, and latency, across all baseline models. For example, our method eliminates

96% HE-rotations on HEFNet, achieving a 19.26× speedup. In contrast, the Hunter-optimized model still has 51% HE-rotations left behind and achieves only 2.22× speedup compared to the un-pruned baseline. This is because our method is designed to eliminate both outer and inner HE-rotations by synthetically applying the HE-group convolution and sub-block pruning, while the state-of-the-art is solely built upon the fixed structure pruning. Besides, we prune 608 out of 921, 15431 out of 18139, 3412 out of 5083, and 3381 out of 6331 blocks for LeNet-like, VGG-5, HEFNet, and ResNet-20 respectively, which exhibits a similar trend to the reported sparsity ratio in Table 5. For illustration purpose, we also plot some examples of encrypted inference-friendly sparsity patterns of kernels of LeNet-like as shown in Figure 5. The weights after optimization are presented in a binary representation. In each convolutional group, it contains 64 weight kernels with size 5×5. Here kernels associated with the same ciphertext, though belonging to different convolutional groups, exhibit the same sparse pattern (x direction). On the other hand, for kernels within the same convolutional group but across different cts, their sparse patterns are different (y direction). When considering deeper networks requiring bootstrapping, we further compare our method with the SOTA (Lee et al., 2022a) under a similar setting, the baseline latency is improved from 647s to 475s, well demonstrating the effectiveness of our method in accelerating deeper networks.

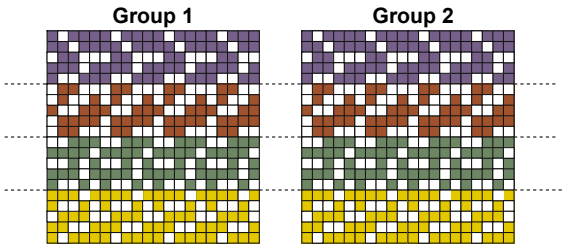

Figure 5. The sparse patterns for weight kernels in LeNet-like 2nd Convolutional layer.

5. Related Work

CryptoNets (Gilad-Bachrach et al., 2016) is an initial attempt to realize HE inference. After that, many subsequent works are proposed to improve the HE-inference latency from different aspects. Faster-CryptoNets (Chou et al., 2018) combines weight pruning and quantization to obtain a sparse polynomial representation to speed up the PMult operation, which achieves 6.38× latency reduction. LoLa (Brutzkus et al., 2019) successfully demonstrates the HE inference on a simple 3-layer model (1 convolutional layer and 2 FC layers) and achieves a 2.2s inference latency on the MNIST sample by leveraging data packing and rotation techniques. Ghodsi et al. (2020); Jha et al. (2021); Mishra et al. (2020); Lou et al. (2020) propose to reduce the cost of non-linear operations in HE-inference since operations like ReLU dominate the latency in the multi-party computation (MPC) setting.

Dathathri et al. (2019); Kim et al. (2022); Lee et al. (2022a); Ran et al. (2022) further refine the row-major coding format to reduce inference latency by packing independent channels in one ciphertext and then performing multi-channel convolution on such packed ciphertexts. However, in these works, the packed channels associated with a ciphertext are still from the same convolution group as that of traditional non-encrypted convolution. As a result, ciphertext outer-rotations cannot be avoided because these channels still need to be added together via outer-rotations to obtain an output channel in encrypted convolution.

Lou & Jiang (2021) propose a neural architecture search (NAS) based method to reduce the number of convolutional layers and then save the encryption parameters to speed up the HE-inference. Kim et al. (2022); Lee et al. (2022a) use a multiplexed packing technique to exploit empty slots caused by average pooling or strided convolution and reduce the use of ciphertexts for acceleration. This work is orthogonal to these methods and can be integrated with them to further improve the latency. Hunter (Cai et al., 2022) attempts to structurally prune the weights to accelerate the HE-inference in the MPC setting. HE-PEx (Aharoni et al., 2022) leverage tile tensor and prune weights by tile sparsity in FC layers. It reduces memory overhead and latency by 60% on the tested autoencoder models in a non-client-aided setting. Different from them, SpENCNN is the first to orchestrate the ciphertext encoding and model sparsity design for HE inference acceleration in a non-client-aided setting, significantly outperforming these works.

6. Acknowledgement

We thank all anonymous reviewers for their constructive comments and suggestions on this work. This work is partially supported by the National Science Foundation (NSF) under Grants No. CNS-2153690, CNS-2247891, CNS-2247892 and CNS-2247893.

7. Conclusion

In this paper, we propose a fast HE-based encrypted inference framework–SpENCNN, which builds upon two novel encryption operation-aware techniques–HE-group convolution and sub-block weight pruning. Experimental results show that our solution can speed up the privacy-preserving inference by 8.37×, 12.11×, 19.26×, and 1.87× on LeNet-like, VGG-5, HEFNet, and ResNet-20, respectively, greatly outperforming the state-of-the-art solutions. In the future, we would like to combine our method with a specific hardware accelerator design to further reduce the total latency and make this solution practical.

References

2016, T. Lenet-like for convolutional mnist model example. https://github.com/tensorflow/models/blob/v1.9.0/tutorials/image/mnist/convolutional.py.

Aharoni, E., Baruch, M., Bose, P., Buyuktosunoglu, A., Drucker, N., Pal, S., Pelleg, T., Sarpatwar, K., Shaul, H., Soceanu, O., et al. He-pex: Efficient machine learning under homomorphic encryption using pruning, permutation and expansion. *arXiv preprint arXiv:2207.03384*, 2022.

Albrecht, M. R., Player, R., and Scott, S. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

Alex, K., Nair, V., and Hinton, G. The cifar-10 dataset, 2014.

Brutzkus, A., Gilad-Bachrach, R., and Elisha, O. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pp. 812–821. PMLR, 2019.

Cai, Y., Zhang, Q., Ning, R., Xin, C., and Wu, H. Hunter: He-friendly structured pruning for efficient privacy-preserving deep learning. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pp. 931–945, 2022.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pp. 409–437. Springer, 2017.

Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. A full rns variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pp. 347–368. Springer, 2019.

Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., and Fei-Fei, L. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

Dathathri, R., Saarikivi, O., Chen, H., Laine, K., Lauter, K., Maleki, S., Musuvathi, M., and Mytkowicz, T. Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 142–156, 2019.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Ghodsi, Z., Veldanda, A. K., Reagen, B., and Garg, S. Cryptonas: Private inference on a relu budget. *Advances in Neural Information Processing Systems*, 33:16961–16971, 2020.

Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pp. 201–210. PMLR, 2016.

Halevi, S. and Shoup, V. Algorithms in helib. In *Annual Cryptology Conference*, pp. 554–571. Springer, 2014.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Idelbayev, Y. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 20xx-xx-xx.

Ioannou, Y., Robertson, D., Cipolla, R., and Criminisi, A. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1231–1240, 2017.

Jha, N. K., Ghodsi, Z., Garg, S., and Reagen, B. Deepreduce: Relu reduction for fast private inference. In *International Conference on Machine Learning*, pp. 4839–4849. PMLR, 2021.

Jung, W., Lee, E., Kim, S., Kim, J., Kim, N., Lee, K., Min, C., Cheon, J. H., and Ahn, J. H. Accelerating fully homomorphic encryption through architecture-centric analysis and optimization. *IEEE Access*, 9:98772–98789, 2021.

Kim, M., Jiang, X., Lauter, K., Ismayilzada, E., and Shams, S. Secure human action recognition by encrypted neural network inference. *Nature communications*, 13(1):1–13, 2022.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. 2012 advances in neural information processing systems (nips). *Neural Information Processing Systems Foundation, La Jolla, CA*, 2012.

Lee, E., Lee, J.-W., Lee, J., Kim, Y.-S., Kim, Y., No, J.-S., and Choi, W. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pp. 12403–12422. PMLR, 2022a.

Lee, J.-W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.-S., et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10: 30039–30054, 2022b.

Li, T., Wu, B., Yang, Y., Fan, Y., Zhang, Y., and Liu, W. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3977–3986, 2019.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations*.

Lou, Q. and Jiang, L. Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International conference on machine learning*, pp. 7102–7110. PMLR, 2021.

Lou, Q., Bian, S., and Jiang, L. Autoprivacy: Automated layer-wise parameter selection for secure neural network inference. *Advances in Neural Information Processing Systems*, 33:8638–8647, 2020.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. Cloud computing—the business perspective. *Decision support systems*, 51(1):176–189, 2011.

Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., and Popa, R. A. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522, 2020.

Peng, H., Zhou, S., Luo, Y., Duan, S., Xu, N., Ran, R., Huang, S., Wang, C., Geng, T., Li, A., et al. Polympcnet: Towards relu-free neural architecture search in two-party computation based private inference. *arXiv preprint arXiv:2209.09424*, 2022.

Ran, R., Wang, W., Gang, Q., Yin, J., Xu, N., and Wen, W. Cryptogcn: Fast and scalable homomorphically encrypted graph convolutional network inference. *Advances in Neural Information Processing Systems*, 35:37676–37689, 2022.

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1xSperKvH.

Rivest, R. L., Adleman, L., Dertouzos, M. L., et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

Samardzic, N., Feldmann, A., Krastev, A., Manohar, N., Genise, N., Devadas, S., Eldefrawy, K., Peikert, C., and Sanchez, D. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *ISCA*, pp. 173–187, 2022.

SEAL. Microsoft SEAL (release 3.4). https://github.com/Microsoft/SEAL, October 2019. Microsoft Research, Redmond, WA.

Smart, N. P. and Vercauteren, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pp. 420–443. Springer, 2010.

Varghese, B. and Buyya, R. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849–861, 2018.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.