RESEARCH-ARTICLE

Hunter: HE-Friendly Structured Pruning for Efficient Privacy-Preserving Deep Learning

YIFEI CAI, Old Dominion University, Norfolk, VA, United States

QIAO ZHANG, Old Dominion University, Norfolk, VA, United States

RUI NING, Old Dominion University, Norfolk, VA, United States

CHUNSHENG XIN, Old Dominion University, Norfolk, VA, United States

HONGYI WU, Old Dominion University, Norfolk, VA, United States

Published: 30 May 2022

ASIA CCS '22: ACM Asia Conference on Computer and Communications Security
May 30 - June 3, 2022
Nagasaki, Japan

Conference Sponsors:
SIGSAC

ASIA CCS '22: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (May 2022)
https://doi.org/10.1145/3488932.3517401
ISBN: 9781450391405

Hunter: HE-Friendly Structured Pruning for Efficient Privacy-Preserving Deep Learning

Yifei Cai, Qiao Zhang, Rui Ning, Chunsheng Xin, Hongyi Wu
ycai001@odu.edu,qzhan002@odu.edu,rning@odu.edu,cxin@odu.edu,h1wu@odu.edu
Old Dominion University
Norfolk, VA, USA

ABSTRACT

In order to protect user privacy in Machine Learning as a Service (MLaaS), a series of ingeniously designed privacy-preserving frameworks have been proposed. The state-of-the-art approaches adopt Homomorphic Encryption (HE) for linear function and Garbled Circuits (GC)/Oblivious Transfer (OT) for nonlinear operation to improve computation efficiency. Despite the encouraging progress, the computation cost is still too high for practical applications. This work represents the first step to effectively prune privacy-preserving deep learning models to reduce computation complexity. Although model pruning has been discussed extensively in the machine learning community, directly applying the plaintext model pruning schemes offers little help to reduce the computation in privacy-preserving models. In this paper we propose *Hunter*, a structured pruning method that identifies three novel HE-friendly structures, i.e., *internal structure*, *external structure*, and *weight diagonal* to guide the pruning process. Hunter outputs a pruned model that, without any loss in model accuracy, achieves a significant reduction in HE operations (and thus the overall computation cost) in the privacy-preserving MLaaS. We apply Hunter in various deep learning models, e.g., AlexNet, VGG and ResNet over classic datasets including MNIST, CIFAR-10 and ImageNet. The experimental results demonstrate that, without accuracy loss, Hunter efficiently prunes the original networks to reduce the HE Perm, Mult, and Add operations. For example, in the state-of-the-art VGG-16 on ImageNet with 10 chosen classes, the total number of Perm is reduced to as low as 2% of the original network, and at the same time, Mult and Add are reduced to only 14%, enabling a significantly more computation-efficient privacy-preserving MLaaS.

CCS CONCEPTS

• **Security and privacy** → *Privacy-preserving protocols.*

KEYWORDS

Model Pruning, Machine Learning as a Service, Privacy-preserving Computation, Homomorphic Encryption

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan.

ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3517401

ACM Reference Format:
Yifei Cai, Qiao Zhang, Rui Ning, Chunsheng Xin, Hongyi Wu. 2022. Hunter: HE-Friendly Structured Pruning for Efficient Privacy-Preserving Deep Learning. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3517401

1 INTRODUCTION

From Amazon's Alexa to Tesla's Model 3 and from Google's AlphaGo to Boston Dynamics's Atlas, Deep Learning (DL) is playing a game-changing role in our daily lives and work. The prevalent and pervasive adoption of DL technology lies in its superior performance to mine the hidden pattern from enormous data [38]. At the same time, the needs to obtain and process such a massive amount of data pose a challenge to many resource-limited individual entities such as a local health provider which intends to build a comprehensive DL model to facilitate diagnoses and healthcare planing, but has limited medical data, computation resources, and DL talents. On the other hand, the technology giants such as Google has abundant cloud data, computation power, and top DL engineers, making them an ideal party to produce well-trained DL models to serve the aforementioned resourced-limited individuals. To bridge this gap, the Machine Learning as a Service (MLaaS) has been proposed [44], where the client, e.g., a doctor in a local clinic, sends the private data, e.g., medical records of her patients, to the server that owns a well-trained DL model; then the server outputs and sends back the prediction to the client. MLaaS provides an efficient solution for the client to obtain cost-effective, high-quality predictions and for the server to make revenue by offering such service.

Challenges in Privacy-Preserving MLaaS: The privacy has become as a critical concern in MLaaS. On the one hand, the client does not want any party including the server to know its private input, e.g., the patient's medical records, and the server does not want to share its proprietary model parameters since training a well-performed DL model involves a significant effort including hardware investment and algorithm design. On the other hand, there is already legislation to protect the data from disclosing to the public such as the Health Insurance Portability and Accountability Act (HIPAA) in the US, the General Data Protection Regulation (GDPR) in EU, and the Personal Data Protection Act (PDPA) in Singapore. There is an urgent need to ensure that the client's data is blind to the server while the server's model parameters are hidden from the client during the interaction in MLaaS.

In order to address the critical privacy issue discussed above, the privacy-preserving MLaaS strategically introduces and embeds crypto primitives into the computation process of the DL model. To this end, a series of ingeniously designed privacy-preserving

frameworks have made inspiring efforts to bring MLaaS into practice [3, 8, 11, 17, 18, 21, 23, 26–29, 31–36, 41–43, 46, 47] where the most commonly adopted crypto primitives are Homomorphic Encryption (HE) [9], Garbled Circuits (GC) [2], Oblivious Transfer (OT) [6], and Secret Sharing (SS) [39]. Among these crypto primitives, the HE is more efficient for linear computation as it intrinsically supports linear functions [4, 9] (see details in Sec. 2.4), while the GC and OT are more computationally-efficient for nonlinear functions [8]. Since the combination of linear and nonlinear functions repeats in the DL model (see details in Sec. 2.1), the privacy-preserving DL frameworks usually adopt HE for linear and GC/OT for nonlinear operations to streamline the privacy-preserving computation [8, 18, 26, 33]. For example, the HE-GC-based frameworks, GAZELLE [18] and DELPHI [26], and the HE-OT-based framework, CrypTFlow2 [33], have achieved a computation speedup of several orders of magnitude over the classic CryptoNets system [11].

Despite the encouraging progress to boost the computation efficiency of privacy-preserving MLaaS, the overhead is still too high for practical applications. For instance, inferring one single CIFAR-10 image [19] over ResNet [15] by the state-of-the-art privacy-preserving frameworks (such as GAZELLE, DELPHI and CrypTFlow2) costs about 100 seconds [25, 47], while many real-time applications require a response within a few seconds [1]. For deeper models with larger inputs, the performance gap would be grow even wider. Meanwhile, the HE-based linear computation takes over 90% of the total time in the above three leading frameworks [47], motivating a deep optimization of the HE-based linear computation to reduce the overall running time in the privacy-preserving MLaaS.

Preliminary Observations: Our quest begins with three insights into the current privacy-preserving MLaaS frameworks. First, nearly all frameworks including GAZELLE, DELPHI and CrypTFlow2 target at the computation optimization over the given DL models such as VGG and ResNet. However, the underlying model redundancy may become the bottleneck to minimize the computation overhead. When a classical model is applied to a given application, many model parameters can be removed for better computation efficiency without any loss in model accuracy [14, 24]. Therefore, our first insight is to prune the DL model to reduce the computation cost in the privacy-preserving MLaaS.

Second, the computation overhead of HE-based calculations stems from high computation complexity of three basic HE operations, i.e., Add, Mult and Perm (see details in Section 2.4). For example, in GAZELLE, the VGG-16 with CIFAR-10 dataset involves about 423K Perm, 7M Mult and 7M Add operations (see detailed performance in Section 4). Therefore, reducing the computation overhead for the HE-based linear calculation is intrinsically to reduce the corresponding number of HE operations. Meanwhile, the Perm operation is the most computation-expensive one among the three HE operations. Experiments show that one Perm is 34 times slower than one Mult and 56 times slower than one Add[1]. As such our second insight is to minimize the number of Perm (as well as Mult and Add) operations while performing model pruning.

Third, the model pruning has been discussed extensively in the machine learning community [14, 24, 45]. The basic idea of model pruning is to first set selected model parameters, e.g., parameters below a threshold [14], to zero, and then retrain or finetune the pruned model to recover accuracy. The above two steps are repeated until the model is maximally pruned with no or negligible accuracy loss. Unfortunately, directly applying the plaintext model pruning offers little to no help to reduce the corresponding HE-based computation over the pruned model (see Table 1). This is because the three basic HE operations (over the ciphertext) work in a packed manner for the linear computation between the input data and the model parameters (see details in Section 2.4). For example, for the element-wise multiplication between one ciphertext encrypted by the client and one plaintext with vectorized weight values from the server, the subsequent Perm operation over the multiplied ciphertext is eliminated *if and only if* all the elements in that plaintext are pruned. However, the plaintext model pruning schemes do not consider such packed structures and rarely guarantee the above desired pruning property, thus leading to marginal or no reduction in the corresponding HE-based computation even though the model is significantly pruned. For example, experiments show that even 65% of parameters in a convolution layer from AlexNet are pruned via the well-known pruning algorithm [14], it only results in about 3.6% of reduction in the Perm operations in the corresponding HE-based computation. Worse yet, pruning 90.8% weights in a fully-connected layer would not even reduce a single Perm out of the total 4096 Perm operations. It is fundamentally a new and nontrivial problem to redesign the model pruning strategies for privacy-preserving MLaaS.

Our Contributions: In this paper, we take the first step to effectively prune privacy-preserving DL models, aiming to significantly reduce the computation cost for the privacy-preserving MLaaS. The proposed framework, *Hunter*, features an HE-friendly, structured pruning method that first identifies the packed structures associated with the Perm operations in the HE-based linear computation and then defines three novel HE-friendly structures, i.e., *internal structure*, *external structure*, and *weight diagonal* that are embedded into a customized pruning process. Hunter outputs a pruned model that, without any loss in model accuracy, achieves to a significant reduction in HE operations (and thus the overall computation cost) in the privacy-preserving MLaaS. For example, we apply Hunter in various DL models including AlexNet [20], VGG [40] and ResNet [15] over classic datasets such as MNIST, CIFAR-10 and ImageNet [19, 22, 37]. The experimental results demonstrate that Hunter effectively reduces the Perm as well as Mult and Add operations in the HE-based linear computation, which contributes to a more computation-efficient privacy-preserving MLaaS. Specifically, Hunter reduces 86% Perm, 84% Mult, and 84% Add operations in average compared to the state-of-the-art frameworks (see detailed performance in Section 4).

Note that there is another research thrust called privacy-preserving Neural Architecture Search (NAS) which, similar to Hunter's privacy-preserving model pruning, aims to find a network structure that is more efficient for privacy-preserving MLaaS using NAS technology [10, 16, 25]. The current methods aim to either replace some nonlinear functions with more computation-efficient ones [10, 16] or search for the optimal crypto parameters, e.g., number of slots and ciphertext/plaintext modulus of packed HE [25]. It is worth pointing out that these approaches are orthogonal to Hunter's

[1] https://github.com/chiraag/gazelle_mpc

pruning method. Hunter can be implemented on top of the NAS-based approaches to further improve its computation efficiency of privacy-preserving MLaaS.

The rest of the paper is organized as follows. Section 2 introduces the system model, threat model, and cryptographic tools adopted in Hunter. The details of Hunter's pruning schemes are elaborated in Section 3. The experimental results are illustrated in Section 4. Finally, Section 5 concludes the paper.

2 PRELIMINARIES

2.1 System Model

In this paper, we consider the MLaaS as shown in Figure 1. There are two parties, i.e., the client C and server $\mathcal{S}$. The former owns sensitive data, e.g., medical records from the healthcare provider, while the latter has a well-performed and usually proprietary DL model to provide the prediction output to the C after receiving C's input. The server $\mathcal{S}$ prunes the DL model before making it available to the clients. This phase is totally client-independent and does not require the sensitive input from C. On the other hand, privacy issues are raised in the interaction between the two parties. Specifically, C does not want any other party including $\mathcal{S}$ to know its private data while $\mathcal{S}$ is unwilling to make its model parameters, e.g., weight and kernel values, public since training that model involves noticeable human and hardware resources. As such, the MLaaS aims to guarantee that C's input is fully protected against the server while $\mathcal{S}$'s model parameters are totally blind to the client.

We focus on deep Convolutional Neural Networks (CNN), which are prevalent and pervasive in many applications such as image classification [20, 40] and face recognition [38]. The convolution and dot product are two main linear functions in CNN. The convolution computation is visualized as placing the kernel at each location of the input and then summing up the element-wise product between the kernel values and the ones of input data within the kernel window, while the dot product is calculated between a weight matrix and a vector such that each output value is the sum of the element-wise product between one row of the weight matrix and that vector. In MLaaS, the kernels and weight matrices are at $\mathcal{S}$ while the input (in the convolution computation) and the vector (in the dot product computation) are from C. As for the nonlinear function, we mainly adopt ReLU, $f(x) = \max\{0, x\}$, which is widely applied in the state-of-the-art DL models such as AlexNet [20], VGG [40] and ResNet [15]. A layer that includes the convolution function is named as a convolution layer while the one with dot product is called a fully-connected (dense) layer. Furthermore, there is another type of layer named pooling, which usually follows after the convolution layer and adopts either meanpooing or maxpooling function. Given the output of the convolution layer, meanpooling

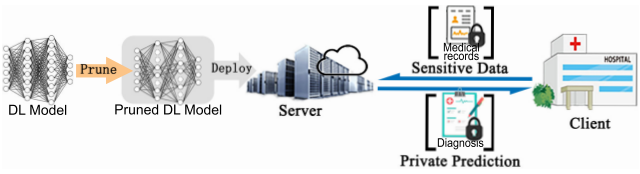

Figure 1: MLaaS with a Model Pruning Phase.

puts a (stridden) pooling window through locations of that output and calculates the mean of the values within the pooling window for each location, while the maxpooling picks the maximum within that pooling window. Note that this paper mainly addresses the optimization for (privacy-preserving) computation efficiency of linear computation , while following the efficient (privacy-preserving) nonlinear calculation including maxpooling in the state-of-the-art frameworks such as GAZELLE [18] and CrypTFlow2 [33].

2.2 Model Pruning

As mentioned earlier, the server trains and prunes a DL model before it provides the MLaaS service and the model largely determines the computation complexity of MLaaS. There are generally two types of pruning namely non-structured pruning [12, 13] and structured pruning [24, 45]. The former prunes the model parameters in kernels and weight matrices that are smaller than a threshold, while the latter removes certain kernels, filters, and even layers [24, 45] as a whole. The existing pruning approaches aim to improve the efficiency of planintext models, i.e, without privacy-preserving consideration. As analyzed in Sections 1 and 3, they offer little help for reducing the computation cost in privacy-preserving MLaaS since they do not consider the computation properties involved in privacy-preserving MLaaS, such as the data encrypted with packed HE and the corresponding element-wise addition, multiplication and permutation operations.

The proposed Hunter is a first-of-its-kind pruning framework, which fully considers the computation in privacy-preserving MLaaS. It analyzes the intrinsic performance bottleneck and identifies three privacy-preserving friendly structures, i.e., *internal structure*, *external structure* and *weight diagonal* (see details in Section 3), for model pruning, which significantly boosts the computation efficiency of MLaaS (see concrete performance results in Section 4).

2.3 Thread Model

In line with many state-of-the-art frameworks such as MiniONN [23], GAZELLE [18], DELPHI [26] and CrypTFlow2 [33], Hunter adopts the semi-honest adversary model where all parties, i.e., C and $\mathcal{S}$, follow the protocol while each party tries to infer extra information from the received messages. Specifically, C tries to infer the model parameters beyond the prediction result, i.e., values of kernels and weight matrices, during the MLaaS interaction, while $\mathcal{S}$ tries to figure out C's private input. We analyze in Section 3.3 that Hunter is secure under the semi-honest assumption.

2.4 Packed Homomorphic Encryption

Homomorphic Encryption (HE) is a kind of encryption that allows linear computation over ciphertext without decryption, yielding a result in the encrypted form of the corresponding plaintext result. Given this critical feature, HE is a widely adopted approach in privacy-preserving MLaaS since the client C is able to encrypt its private data and send it to the $\mathcal{S}$, which conducts the computation over C's ciphertext based on its DL model, without knowing the C's secrets. Meanwhile, compared with the traditional HE algorithm that individually encrypts each value [30], the packed HE directly encrypts a vector of plaintext values into one ciphertext and performs the computation over that ciphertext in a Single Instruction

Multiple Data (SIMD) manner [5]. Therefore, it is highly efficient and widely adopted in the state-of-the-art MLaaS frameworks [44]. In this paper, we rely on the BFV algorithm [9], which is one of the mainstream packed HE approaches. On the one hand, the packed HE supports three basic linear operations as Homomorphic Addition (Add), Homomorphic Multiplication (Mult), and Homomorphic Permutation (Perm). Specifically, given two plaintext m-value vectors v_1 and v_2, they are respectively encrypted by packed HE as $[v_1]_C$ and $[v_2]_C$. Thereafter, we denote $[\cdot]_C$ and $[\cdot]_S$ as the ciphertext encrypted by the C and the S, respectively. The Add operation produces another ciphertext $[v_1 + v_2]_C$ that adds the $[v_1]_C$ and $[v_2]_C$ in an element-wise manner. The Mult operation outputs the ciphertext $[v_1 \odot v_2]_C$ that performs element-wise multiplication between the $[v_1]_C$ and $[v_2]_C$. The Perm operation is a cyclic rotation of the values in one ciphertext. For example, rotating $[v_1]_C$ results in another ciphertext where the value at the i-th position moves to the first position. Meanwhile, performing j Perm operations over one ciphertext can be decomposed into one PerDecomp operation and j HstPerm operations [18], which contributes to a lower amortized operation time for that ciphertext.

The proposed Hunter framework features with a significant reduction for HE operations including Perm and thus also contributes to the reduction for PerDecomp and HstPerm operations. On the other hand, the Perm operation has the highest running-time complexity among the three basic HE operations. Concretely, our experiment shows that the running time of one Perm operation is 56 times slower than one Add and 34 times slower than one Mult. As the Perm operation is indispensable and imperative for the HE-based linear computation, i.e., convolution and dot product, Hunter aims to minimize the computation complexity of Perm and thus correspondingly reduces the HE-based computation in privacy-preserving MLaaS. Meanwhile, the model pruning in the machine learning community seems to be a good solution to reduce the computation overhead, but it dose not consider the packed nature of HE operations and thus has little contribution to reduce the complexity of HE-based computation. Specifically, for the multiplied ciphertext between the encrypted input from the client and the weight plaintext from the server, one subsequent Perm operation is eliminated if and only if all values in that weight plaintext are pruned. Hunter fully considers such packed feature in HE-based computation, and proposes, for the first time, to train and prune a DL model with HE-friendly pruning strategy and finally outputs a pruned model that significantly reduces computation complexity.

3 SYSTEM DESCRIPTION

Privacy-preserving DL outputs a prediction to the client with both the client's input data and the server's model parameters protected during the computation process. A critical issue in this scenario is the computation overhead, which hinders its adoption in many real-world applications. A series of works have been done to consistently improve the system performance of privacy-preserving DL [3, 8, 11, 17, 18, 21, 23, 26–29, 31–36, 41–43, 46, 47]. Among them, the hybrid schemes that respectively utilize cryptographic tools for linear and nonlinear functions achieve better performance [8]. The (packed) HE algorithms, e.g., BFV [9] and CKKS [7], are mainly adopted in the hybrid schemes for efficient linear computation.

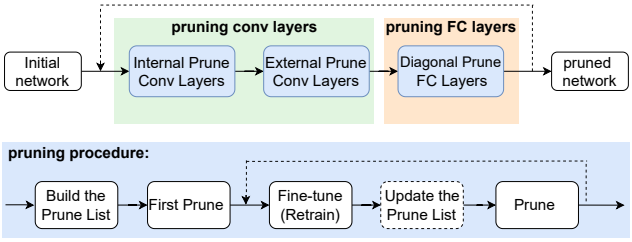

Figure 2: Overview of Hunter's Pruning Approach.

However, the HE-based linear computation takes a large part, e.g., over 90% of the total cost in the hybrid privacy-preserving DL frameworks as discussed in Section 1. All of the current solutions [18], [33], [26] focus on how to improve the efficiency of HE-based linear computation given existing DL model structures such as AlexNet [20] and VGG [40]. Considering the noticeable redundancy in these modern DL models, a straightforward takeaway is to first prune the target DL model via plaintext solutions [13], [24], [45], thus reducing the size of the MLaaS model. However, as discussed above, directly applying plaintext pruning mechanism has little effect on computation reduction for corresponding HE-based computation. Therefore, a new approach is needed to make the model pruning truly helpful for reducing HE-based linear computation and thus contributing to significant efficiency improvement of privacy-preserving MLaaS.

In this work, we propose Hunter, an HE-friendly structured pruning for efficient privacy-preserving deep learning. Without accuracy loss, Hunter targets at pruning the model parameters, i.e., weights and kernels, to essentially reduce the three basic operations over the client C's encrypted inputs or intermediate ciphertext, including Mult, Add and especially Perm (which is the most expensive operation in HE-based linear computation). Figure 2 shows an overall design of Hunter's pruning mechanism. By identifying and defining three HE-friendly structures in HE-based linear computation, i.e., *internal structure*, *external structure*, and *weight diagonal*, three pruning modules, i.e., internal pruning and external pruning for convolution, and diagonal pruning for dot product, are correspondingly and carefully embedded into the whole pruning process, and a pruned model is finally formed which enables aggressive elimination of involved Perm (as well as the associated HE additions and multiplications) and thus significantly reduces the overall computation cost. Experiments show that Hunter has a promising performance that reduces 94%, 81%, 79%, 86% and 49% of Perm operations on AlexNet, VGG-11, VGG-13, VGG-16 and ResNet-32 with CIFAR-10 dataset, and 55% of Perm operations on LeNet with MNIST dataset, as well as 64% and 98% of Perm operations on AlexNet and VGG-16 with ImageNet dataset (with 10 classes). To the best of our knowledge, Hunter is the first framework that defines the (packed) HE-friendly structures for loss-free pruning, and makes the model pruning truly useful to privacy-preserving DL computation. Hunter may also shed light on efficient privacy-preserving friendly pruning and inspire subsequent research works that further close the gap to practical privacy-preserving DL. In the following subsections, we describe in details Hunter's underlying pruning logic from dot product to convolution computation.

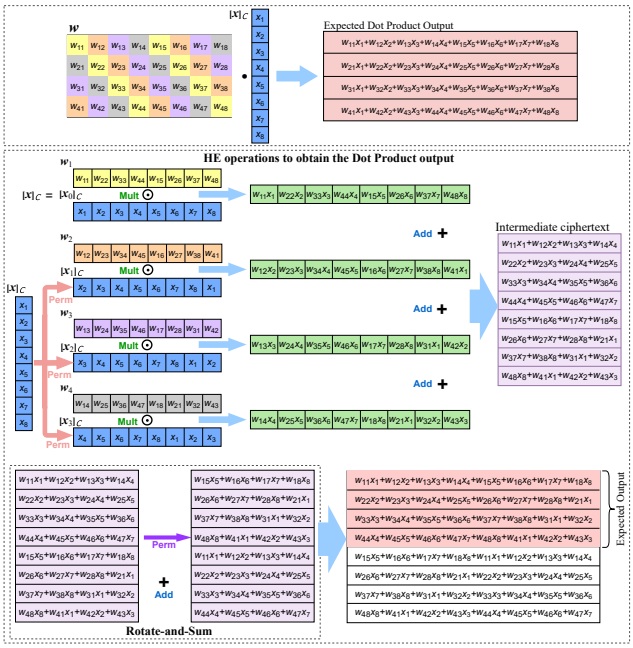

Figure 3: Dot Product Computation.

3.1 HE-Friendly Structured Pruning for Dot Product Computation

As described in Section 2.1, the convolution (in the convolution layer) and dot product (in the fully-connected layer) are two main types of linear computation in the DL model. For ease of elaboration, we begin with the state-of-the-art computation and Hunter's pruning insights for dot product, which is intrinsically a matrix-vector multiplication between the client-encrypted vector $[\boldsymbol{x}]_C$ and the plaintext weight matrix $\boldsymbol{w}$ at the server. Specifically, $\boldsymbol{x}$ is an n_i-element vector that is packed into one ciphertext as $[\boldsymbol{x}]_C$ and $\boldsymbol{w}$ has a size of $n_o \times n_i$. The server is supposed to compute the dot product based on $[\boldsymbol{x}]_C$ and its plintext $\boldsymbol{w}$, and output one ciphertext containing n_o elements of the plaintext dot product between $\boldsymbol{w}$ and $\boldsymbol{x}$.

Recall from Section 2.1 that the dot product is calculated between the weight matrix $\boldsymbol{w}$ and the input vector $\boldsymbol{x}$ such that each output value is the sum of the element-wise product between one row of $\boldsymbol{w}$ and $\boldsymbol{x}$ (see the upper part of dot product computation in Figure 3). Under the use of packed HE in the privacy-preserving DL frameworks [18, 26, 33], the element-wise product is computed between $[\boldsymbol{x}]_C$ and each row of $\boldsymbol{w}$ by the Mult operation. However, it is not possible for the server to directly sum up the element-wise product in the above multiplied ciphertext since the Add operation also works in element wise (see details in Sec. 2.4). Therefore, the Perm operation is needed to efficiently complete the summing-up step to finally output the dot product result [18, 33].

Specifically, n_o plaintext vectors, each of which contains a diagonal of the weight matrix, are constructed by the server (see the color-coded diagonals of $\boldsymbol{w}$ in Figure 3 where we have $n_i = 8, n_o = 4$). Then, $(n_o - 1)$ Perm operations are conduct over the input ciphertext[2] $[\boldsymbol{x}]_C$ such that the j-th $(1 \leqslant j \leqslant n_o)$ of these n_o rotated ciphertext (including the original $[\boldsymbol{x}]_C$) has the same sequence of associated weight values as that of the j-th of n_o plaintext vectors. For example, there are four plaintext vectors in Figure 3 as $\boldsymbol{w}_1, \boldsymbol{w}_2$, $\boldsymbol{w}_3$ and $\boldsymbol{w}_4$. Here $\boldsymbol{w}_1$ includes weight values from w_{11} to w_{48}, $\boldsymbol{w}_2$ includes weight values from w_{12} to w_{41}, $\boldsymbol{w}_3$ includes weight values from w_{13} to w_{42} and $\boldsymbol{w}_4$ includes weight values from w_{14} to w_{43}. Three Perm operations are performed over $[\boldsymbol{x}]_C$ to obtain four rotated ciphertext (including the original $[\boldsymbol{x}]_C$) as $[\boldsymbol{x}_0]_C$, $[\boldsymbol{x}_1]_C$, $[\boldsymbol{x}_2]_C$ and $[\boldsymbol{x}_3]_C$. Here $[\boldsymbol{x}_0]_C$ has the same sequence of associated weight values as that of $\boldsymbol{w}_1$, $[\boldsymbol{x}_1]_C$ has the same sequence of associated weight values as that of $\boldsymbol{w}_2$, and so on and so forth.

The reason for conducting $(n_o - 1)$ Perm operations over $[\boldsymbol{x}]_C$ is that the n_o rotated ciphertext are directly multiplied with the n_o plaintext vectors (by Mult) to get n_o multiplied ciphertext that have the same sequence of associated element-wise product of the output, and they are further added (by Add) to form the intermediate ciphertext (see the intermediate ciphertext in Figure 3), which can efficiently derive the final dot product result by a series of "Rotate-and-Sum" (RaS) operations [18]: first rotate the intermediate ciphertext by $\frac{n_i}{2}$ positions via Perm and then add the two ciphertext before and after the rotation via Add. It results in a new ciphertext whose first $\frac{n_i}{2}$ elements are the sum of the first and second halves of original intermediate ciphertext. By repeating this process for $\log_2 \frac{n_i}{n_o}$ iterations, each of which rotates by half the previous rotation positions (i.e, $\frac{n_i}{4}$, $\frac{n_i}{8}$, ..., n_o.), we finally get a ciphertext whose first n_o elements are the expected dot product output. For example, the four multiplied ciphertext in Figure 3 have the same sequence of associated element-wise product of the output (from the first to the fourth value of the dot product result), thus, they are added to form an intermediate ciphertext, over which one RaS operation is performed to get the final dot product result.

Observation 1: Each of the n_o multiplied ciphertext is obtained by multiplying one of the n_o plaintext vectors with one of the n_o rotated ciphertext (obtained by Perm). These n_o multiplied ciphertext are added to get the intermediate ciphertext, over which the final dot product result is derived by $\log_2 \frac{n_i}{n_o}$ RaS operations (including $\log_2 \frac{n_i}{n_o}$ Perm operations). Meanwhile, the Perm operations in RaS computation are inevitable as we cannot eliminate any intermediate ciphertext.

It is critical to observe that due to the unique computation structure specially designed for the n_o multiplied ciphertext, unless a plaintext vector becomes all-zero, the state-of-the-art algorithm would still require the Perm operation over the input ciphertext to get that multiplied ciphertext. While the pruning technique is a good solution to possibly eliminate the Perm operations, the traditional plaintext schemes aim to prune individual weights and do not consider the specific structure of each of the n_o plaintext vectors, which helps little for the Perm reduction (and thus the efficient HE-based computation) as discussed in Section 1.

[2] As multiple Perm operations over one ciphertext can be decomposed into one PerDecomp operation and the equal amount of HstPerm operations for more efficient computation [18], the $(n_o - 1)$ Perm operations over $[\boldsymbol{x}]_C$ can be decomposed into one PerDecomp operation and $(n_o - 1)$ HstPerm operations.

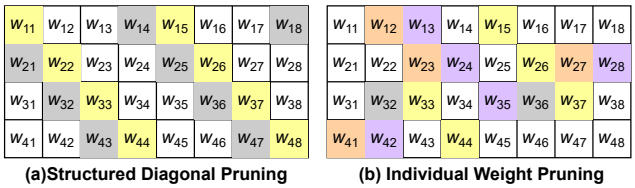

Figure 4: Two pruning methods for fully-connected layers.

Thus, different from the plaintext pruning schemes, *we are motivated to prune the diagonal weights in the plaintext vectors such that there is no need to form the rotated ciphertext (by Perm) to be multiplied with the pruned plaintext vectors.* Therefore, one Perm operation is eliminated if one plaintext vector is pruned, which helps for the Perm reduction and thus the efficient HE-based computation. For example, in Figure 3, one Perm operation over $[\boldsymbol{x}]_C$ (to obtain $[\boldsymbol{x}_3]_C$) is eliminated by pruning the plaintext vector $\boldsymbol{w}_4$. Moreover, other subsequent operations related to the pruned plaintext vectors are also simultaneously eliminated.

Hunter's diagonal pruning structure: Based on the above observation, we define the *weight diagonal* as the diagonal elements in one plaintext vector (excluding the ones in the main-diagonal plaintext vector) and propose Hunter's *diagonal pruning* strategy by pruning the *weight diagonals*, which efficiently reduces the Perm operations in HE-based dot product computation.

Figure 4 visually compares the pruning schemes between Hunter's diagonal pruning and the traditional plaintext pruning [14] for a weight matrix with the size of 4×8. While Hunter's diagonal pruning (in Figure 4 (a)) removes two *weight diagonals* (colored in white), which correspondingly reduces two (out of three) Perm operations, the traditional plaintext pruning (in Figure 4 (b)) does not eliminate any Perm operations even with the same pruning ratio. The reason lies in Hunter's HE-friendly pruning structure (i.e., the *weight diagonal*) identified and defined in the dot product computation, which is not considered in the traditional plaintext pruning. Furthermore, our experiments also demonstrate that, without sacrificing model accuracy, Hunter effectively reduces 65% of Perm operations for the dot product computation with a 4096×4096 weight matrix in one fully-connected layer of AlexNet with ImageNet dataset, while the traditional threshold-based pruning algorithm cannot eliminate any of the involved 4095 Perm operations (see details in Table 7 from Appendix A).

3.2 HE-Friendly Structured Pruning for Convolution Computation

While the above discussions shed light on pruning the weight matrices for fully-connected layers in privacy-preserving DL models, the modern DL architectures, such as AlexNet [20] and VGG [40], are dominated by convolution layers. It is thus critical to effectively prune the kernels for convolution computation and therefore efficiently reduce the Perm operations in convolution layers. As such, we begin with the basic HE-based convolution computation with Single Input and Single Output (SISO) where we identify an HE-friendly pruning primitive. Then we move on to the general convolution computation with Multiple Input Multiple Output (MIMO)

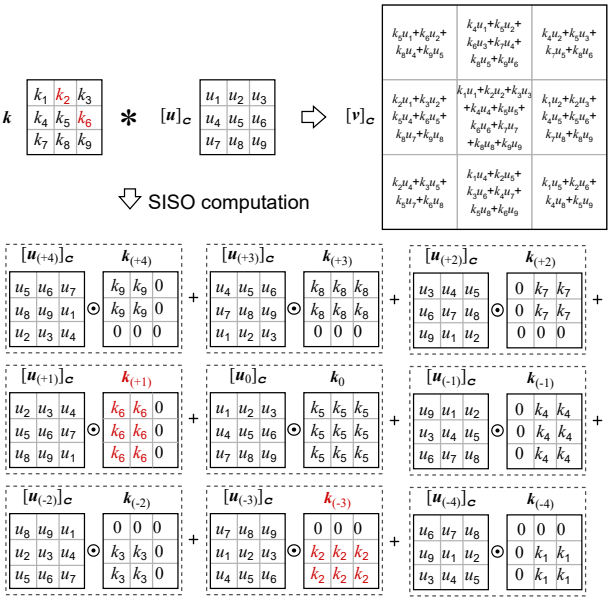

Figure 5: Basic Pruning for SISO.

(proposed in the state-of-the-art GAZELLE framework [18]) where we define the *internal structure* based on the primitive from SISO, as well as the *external structure* based on the rotation pattern over intermediate ciphertext. The two defined structures result in aggressively pruned model(s) for efficient HE-based computation without accuracy loss. For example, Hunter reduces 85%, 84%, and 84% of Perm, Mult and Add operations, respectively, for convolution layers in VGG-16 with CIFAR-10 dataset compared with GAZELLE. We denote that the input data has c_i channels, each with a size of $u_w \times u_h$, and there are c_o kernels, each with a size of $k_w \times k_h \times c_i$. In SISO, we have $c_i = c_o = 1$ while c_i or c_o is larger than one in MIMO.

SISO: As shown in Figure 5, the client C encrypts its input $\boldsymbol{u}$ (with the size of $u_w \times u_h$) as $[\boldsymbol{u}]_C$, which is then sent to the server $\mathcal{S}$. $\mathcal{S}$ conducts convolution computation between $[\boldsymbol{u}]_C$ and its plaintext kernel $\boldsymbol{k}$ with size of $k_w \times k_h$ to obtain the encrypted output $[\boldsymbol{v}]_C$ where $\boldsymbol{v} = \boldsymbol{k} * \boldsymbol{u}$ and "$*$" is the convolution operator. Specifically, the convolution is calculated by first placing the kernel at each location of the input and then summing up the element-wise products between the kernel values and the ones of input data within the kernel window. For example, we have $u_w = u_h = k_w = k_h = 3$ in Figure 5, and the first value of convolution is obtained by placing the k_5, i.e., the central value of kernel $\boldsymbol{k}$, at u_1, i.e., the first element in $\boldsymbol{u}$, and summing up the element-wise product as $(k_5u_1 + k_6u_2 + k_8u_4 + k_9u_5)$. The second value of convolution is obtained by placing the k_5 at u_2, i.e., the second element in $\boldsymbol{u}$, and summing up the element-wise product as $(k_4u_1 + k_5u_2 + k_6u_3 + k_7u_4 + k_8u_5 + k_9u_6)$, and so on and so forth. As such, nine values are obtained as the final convolution output $\boldsymbol{v}$ for SISO (see Figure 5).

Using packed HE in the privacy-preserving DL frameworks [18, 26, 33], the element-wise product is obtained by multiplying the input ciphertext with the kernel. For example, as illustrated in Figure 5, multiplying $[\boldsymbol{u}]_C$ with $\boldsymbol{k}$ by Mult outputs another ciphertext containing nine elements as $u_1k_1, \cdots, u_9k_9$. However, as discussed

in Sec. 3.1, it is not possible for the server $\mathcal{S}$ to directly sum up the elements in the ciphertext and the Perm is thus needed to complete the summing-up step to let $\mathcal{S}$ finally obtain the encrypted convolution values.

Specifically, the i-th ($1 \leqslant i \leqslant u_w u_h$) value of the convolution output is the sum of element-wise product between the $k_w k_h$ kernel elements and the associated $k_w k_h$ elements around the i-th location of the input, i.e., the $k_w k_h$ input elements within the kernel window. The $(k_w k_h - 1)$ Perm operations[3], including $(k_w k_h - 1)/2$ in forward (left) direction and $(k_w k_h - 1)/2$ in backward (right) direction, are performed over $[\boldsymbol{u}]_C$ such that $k_w k_h$ rotated ciphertext (including the original $[\boldsymbol{u}]_C$) are obtained, and the values from all the i-th locations of $k_w k_h$ (rotated) ciphertext include all the input elements needed to calculate the element-wise product of the i-th convolution value. For example, the fifth element of the convolution output in Figure 5, i.e., the central value, is the the sum of element-wise product between input values from u_1 to u_9 and the kernel values from k_1 to k_9. By conducting four Perm operations in the forward direction, we obtain four rotated ciphertext as $[\boldsymbol{u}_{(+1)}]_C$, $[\boldsymbol{u}_{(+2)}]_C$, $[\boldsymbol{u}_{(+3)}]_C$ and $[\boldsymbol{u}_{(+4)}]_C$, and we similarly get another four rotated ciphertext as $[\boldsymbol{u}_{(-1)}]_C$, $[\boldsymbol{u}_{(-2)}]_C$, $[\boldsymbol{u}_{(-3)}]_C$ and $[\boldsymbol{u}_{(-4)}]_C$ by performing four Perm operations in the backward direction[4]. Together with $[\boldsymbol{u}_0]_C$ (i.e., the original $[\boldsymbol{u}]_C$), the fifth element in each of the above nine ciphertext is associated with one input value (among u_1 to u_9) to calculate the element-wise product for fifth element of the convolution output.

Then Mult and Add operations are conducted (by the server) between the $k_w k_h$ rotated ciphertext and the plaintext kernel to finally get the convolution output. For example, nine (i.e., $k_w k_h$) transformed kernels are constructed in Figure 5 (see $\boldsymbol{k}_{(-4)}$ to $\boldsymbol{k}_{(+4)}$) such that the i-th ($1 \leqslant i \leqslant 9$) value in $\boldsymbol{k}_j$ ($-4 \leqslant j \leqslant +4$) corresponds to the kernel element to be multiplied with the i-th input value in $[\boldsymbol{u}_j]_C$ to form the element-wise product of the i-th convolution value. As such, the i-th value in $\sum_{j=-4}^{+4}([\boldsymbol{u}_j]_C \odot \boldsymbol{k}_j)$, i.e., the summation of nine multiplied ciphertext between $\boldsymbol{u}_j$ and $\boldsymbol{k}_j$, is the i-th element of the convolution output. Furthermore, the cyclic effect of Perm operation on all elements in each $[\boldsymbol{u}_j]_C$, i.e., the rotated ciphertext, makes the values in the corresponding $\boldsymbol{k}_j$, i.e., the transformed kernel, associate with only one element from the original kernel $\boldsymbol{k}$. E.g., the values in $\boldsymbol{k}_0$ (see Figure 5) associate with k_5 from $\boldsymbol{k}$, and the values in $\boldsymbol{k}_{(+1)}$ associate with k_6 from $\boldsymbol{k}$, so on and so forth.

Observation 2: Each rotated ciphertext (obtained by one Perm operation) is multiplied with a transformed kernel that includes only one kernel value, thus one Perm operation for obtaining one rotated ciphertext is eliminated if the kernel value in that to-be-multiplied transformed kernel is zero.

Based on this observation, one Perm operation is eliminated in SISO computation when one kernel value (excluding the central one) in $\boldsymbol{k}$ is zero. Therefore, we are motivated to prune the individual value in $\boldsymbol{k}$ such that there is no need to correspondingly form the rotated ciphertext by one expensive Perm operation. For example,

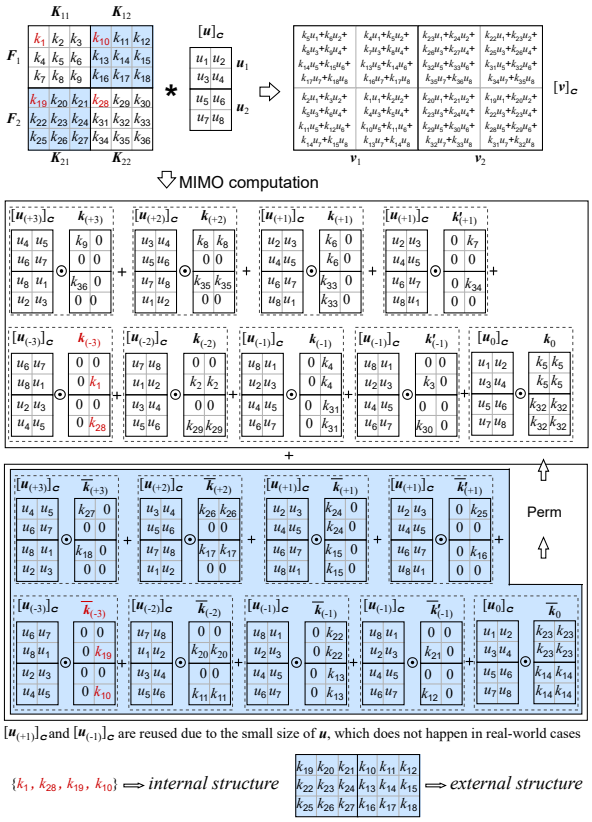

Figure 6: Basic Pruning for MIMO.

as shown in Figure 5, two Perm operations for getting the rotated ciphertext, $[\boldsymbol{u}_{(-3)}]_C$ and $[\boldsymbol{u}_{(+1)}]_C$, are omitted if the kernel values k_2 and k_6 are pruned. This HE-friendly pruning primitive in SISO further motivate our definition of *internal structure* that contributes to the Perm reduction in general MIMO calculation to be described next.

MIMO: In most of the state-of-the-art DL models such as AlexNet and VGG, the convolution computation involves multiple input and multiple output, namely c_i input channels (each with size of $u_w \times u_h$) are convolved with c_o filters (each with size of $k_w \times k_h \times c_i$) and there are c_o output channels with size of $u_w \times u_h$[5]. Under the use of packed HE in privacy-preserving MLaaS [18, 26, 33], c_n input channels are packed in one ciphertext, over which the server homomorphicly computes with its plaintext kernels. For example, we have in Figure 6 two input channels $\{\boldsymbol{u}_1, \boldsymbol{u}_2\}$ in the C-encrypted ciphertext $[\boldsymbol{u}]_C$. It is convolved with two filters, $\boldsymbol{F}_1 = \{\boldsymbol{K}_{11}, \boldsymbol{K}_{12}\}$ and $\boldsymbol{F}_2 = \{\boldsymbol{K}_{21}, \boldsymbol{K}_{22}\}$, each of which has two 3-by-3 kernels. Thus we have $c_i = c_n = 2$, $c_o = 2$, $u_w = u_h = 2$, $k_w = k_h = 3$, and the server is supposed to homomorphicly compute over $[\boldsymbol{u}]_C$ with its two plaintext filters to finally obtain the encrypted convolution $[\boldsymbol{v}]_C$ with two output channels $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ (see Figure 6). Here

$$\boldsymbol{v}_1 = \boldsymbol{K}_{11} * \boldsymbol{u}_1 + \boldsymbol{K}_{12} * \boldsymbol{u}_2 \text{ and } \boldsymbol{v}_2 = \boldsymbol{K}_{21} * \boldsymbol{u}_1 + \boldsymbol{K}_{22} * \boldsymbol{u}_2, \quad (1)$$

where each individual convolution works in SISO manner.

[3] These $(k_w k_h - 1)$ Perm operations can be decomposed into one PerDecomp operation and $(k_w k_h - 1)$ HstPerm operations in GAZELLE framework [18].

[4] Here we use positive and negative symbol "+" and "−" to denote the forward and backward direction, respectively. Similar logic is applied to the subscript of the transformed kernel described later.

[5] Here we consider the same-style convolution with the size of each output channel equals to that of each input channel. Similar logic is applied to the convolution where the size of each output channel is not equal to that of each input channel.

For a lucid description, we elaborate thereafter the state-of-the-art MIMO computation [18] as well as Hunter's pruning insights according to the example in Figure 6. Nevertheless, Hunter is readily applicable to other general cases. Specifically, the C-encrypted ciphertext $[\boldsymbol{u}]_C$ is firstly convolved with the main-diagonal kernels $\{\boldsymbol{K}_{11}, \boldsymbol{K}_{22}\}$ in the SISO manner, which produces a convolved ciphertext containing two convolution as $\boldsymbol{K}_{11} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{22} * \boldsymbol{u}_2$. This is achieved thanks to the cyclic effect of the Perm operation on all elements in $[\boldsymbol{u}]_C$ and when the $\boldsymbol{u}_1$ in $\boldsymbol{u}$ is convolved with one kernel (like SISO), the $\boldsymbol{u}_2$ in $\boldsymbol{u}$ is similarly and simultaneously convolved with another kernel. To this end, Perm operations are firstly applied on $[\boldsymbol{u}]_C$ to get a group of rotated ciphertext (like SISO), which are then multiplied (by Mult) with the transformed-kernel-packs and all the multiplied ciphertext are finally added up (by Add) to get a convolved ciphertext containing two convolution as $\boldsymbol{K}_{11} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{22} * \boldsymbol{u}_2$ (see the upper part of MIMO computation in Figure 6). Clearly, the group of rotated ciphertext for $[\boldsymbol{u}]_C$ (e.g., see $[\boldsymbol{u}_{(+3)}]_C$ to $[\boldsymbol{u}_{(-3)}]_C$) can be used to convolve with any other kernels since we can arbitrarily replace values in $\boldsymbol{K}_{11}$ and $\boldsymbol{K}_{22}$ with other ones. Therefore, another convolved ciphertext containing two convolution as $\boldsymbol{K}_{21} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{12} * \boldsymbol{u}_2$ is obtained for the other diagonal kernels $\{\boldsymbol{K}_{21}, \boldsymbol{K}_{12}\}$ (see the lower part of MIMO computation in Figure 6).

Observation 3: First, based on our observation in SISO, each rotated channel in one rotated ciphertext is multiplied with only one value from one kernel[6]. For example, in the rotated ciphertext $[\boldsymbol{u}_{(-3)}]_C$ in the upper part of MIMO computation, the rotated channel $\boldsymbol{u}_1$ is multiplied with k_1 from kernel $\boldsymbol{K}_{11}$ while the rotated channel $\boldsymbol{u}_2$ is multiplied with k_{28} from kernel $\boldsymbol{K}_{22}$. Second, the cyclic effect of Perm operation makes all the kernel values in each transformed-kernel-pack come from the same location of their corresponding diagonal kernels. For example, the aforementioned values $\{k_1, k_{28}\}$ in the transformed-kernel-pack $\boldsymbol{k}_{(-3)}$ come from the first location of diagonal kernels $\{\boldsymbol{K}_{11}, \boldsymbol{K}_{22}\}$, respectively. Note that while this observation is made using Figure 6 as an example, it holds for the general case with all input sizes and other kernel sizes in real-world DL models.

Based on the above observation, for all the transformed-kernel-packs that are multiplied with one rotated ciphertext, all the kernel values come from the same location of the associated diagonal kernels. Meanwhile, as one rotated ciphertext includes c_n channels ($c_n = 2$ for our example in Figure 6), there must be c_n kernel values (from the same location of the c_n diagonal kernels) in every transformed-kernel-pack (to be multiplied with above rotated ciphertext). For example, two transformed-kernel-packs $\boldsymbol{k}_{(-3)}$ and $\overline{\boldsymbol{k}}_{(-3)}$ are multiplied with the rotated ciphertext $[\boldsymbol{u}_{(-3)}]_C$. Since $[\boldsymbol{u}_{(-3)}]_C$ includes two rotated channels, both of $\boldsymbol{k}_{(-3)}$ and $\overline{\boldsymbol{k}}_{(-3)}$ have two kernel values from the same location of two diagonal kernels, i.e., the first location of two diagonal kernels in $\{\boldsymbol{K}_{11}, \boldsymbol{K}_{22}\}$ and $\{\boldsymbol{K}_{21}, \boldsymbol{K}_{12}\}$, respectively. Therefore, one Perm operation over $[\boldsymbol{u}]_C$ for getting one rotated ciphertext is eliminated if the kernel values from the same location of all the diagonal kernels (to be convolved with the encrypted input $[\boldsymbol{u}]_C$) are zeros. For example, there is no need to get the rotated ciphertext $[\boldsymbol{u}_{(-3)}]_C$ if the kernel values from the first location of all the diagonal kernels (i.e., $\{\boldsymbol{K}_{11}, \boldsymbol{K}_{22}\}$ and $\{\boldsymbol{K}_{12}, \boldsymbol{K}_{21}\}$) are zeros (i.e., kernel values $\{k_1, k_{10}, k_{19}, k_{28}\}$ are zeros).

Hunter's internal pruning structure: As such, we are motivated to prune those same-location kernel values to reduce the number of Perm operations over the encrypted input $[\boldsymbol{u}]_C$ and we thus define the *internal structure* as the kernel values from the same location of all the diagonal kernels that are convolved with $[\boldsymbol{u}]_C$, and pruning an *internal structure* correspondingly eliminates one Perm operation over $[\boldsymbol{u}]_C$. For example, the aforementioned kernel values $\{k_1, k_{10}, k_{19}, k_{28}\}$ actually form one *internal structure* that, if pruned, contributes to eliminate one Perm operation over $[\boldsymbol{u}]_C$.

Recall that we have obtained two (intermediate) convolved ciphertext. The first one contains two convolution as $\boldsymbol{K}_{11} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{22} * \boldsymbol{u}_2$ while the second one has two convolution as $\boldsymbol{K}_{21} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{12} * \boldsymbol{u}_2$. However we are supposed to get a ciphertext including the two output channels $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Directly adding the two convolved ciphertext results in a ciphertext having two convolution as $(\boldsymbol{K}_{11} * \boldsymbol{u}_1 + \boldsymbol{K}_{21} * \boldsymbol{u}_1)$ and $(\boldsymbol{K}_{22} * \boldsymbol{u}_2 + \boldsymbol{K}_{12} * \boldsymbol{u}_2)$, which is not the one as $\boldsymbol{v}_1$ or $\boldsymbol{v}_2$ in Eq. (1). The reason lies in the sequence mismatch of associated filters among the two convolved ciphertext. For example, the first of the two convolved ciphertext has two convolution related to filters $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$, while the second one has two convolution related to the filters $\boldsymbol{F}_2$ and $\boldsymbol{F}_1$. Therefore, all of those convolved ciphertext (excluding the one obtained by convolution between $[\boldsymbol{u}]_C$ and the main-diagonal kernels $\{\boldsymbol{K}_{11}, \boldsymbol{K}_{22}\}$) need to be rotated (by Perm) to have the same sequence of associated filters, and then we can sum up those rotated ciphertext (by Add) to get a ciphertext with the output channels. For example, the second convolved ciphertext (with two convolution as $\boldsymbol{K}_{21} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{12} * \boldsymbol{u}_2$) is rotated by one Perm operation to have two convolution as $\boldsymbol{K}_{12} * \boldsymbol{u}_2$ and $\boldsymbol{K}_{21} * \boldsymbol{u}_1$ (see the Perm operation at the lower part of MIMO computation in Figure 6), and such rotated ciphertext is added (by Add) with the first convolved ciphertext (with two convolution as $\boldsymbol{K}_{11} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{22} * \boldsymbol{u}_2$) to finally get the correct ciphertext with two output channels $\boldsymbol{v}_1 = (\boldsymbol{K}_{11} * \boldsymbol{u}_1 + \boldsymbol{K}_{12} * \boldsymbol{u}_2)$ and $\boldsymbol{v}_2 = (\boldsymbol{K}_{22} * \boldsymbol{u}_2 + \boldsymbol{K}_{21} * \boldsymbol{u}_1)$.

Observation 4: If the diagonal kernels (excluding the main-diagonal kernels), which involve in the convolution with encrypted input $[\boldsymbol{u}]_C$) to get a convolved ciphertext, are zeros, there is no need to further rotate that convolved ciphertext to unify its sequence of associated filters. For example, the Perm operation over the second convolved ciphertext (with two convolution as $\boldsymbol{K}_{21} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{12} * \boldsymbol{u}_2$) is eliminated if the involved diagonal kernels $\{\boldsymbol{K}_{21}, \boldsymbol{K}_{12}\}$ (to get above convolved ciphertext) are all zeros (see the lower part of MIMO computation in Figure 6).

Hunter's external pruning structure: Therefore, we are motivated to prune such diagonal kernels (excluding the main-diagonal kernels) to eliminate the subsequent Perm operation over the convolved ciphertext, and we thus define the *external structure* as these diagonal kernels involved in the convolution with the encrypted input $[\boldsymbol{u}]_C$ for getting a convolved ciphertext, which should be rotated to unify its sequence of associated filters as we have described. Obviously, pruning an *external structure* correspondingly eliminates one Perm over one convolved ciphertext. For instance,

[6]There are no $\boldsymbol{u}_{(+4)}$ and $\boldsymbol{u}_{(-4)}$ in our example since $u_w u_h = (k_w k_h - 1)/2$ and thus we see the reuse of rotated ciphertext $\boldsymbol{u}_{(+1)}$ and $\boldsymbol{u}_{(-1)}$. No rotated ciphertext is reused in other general cases.

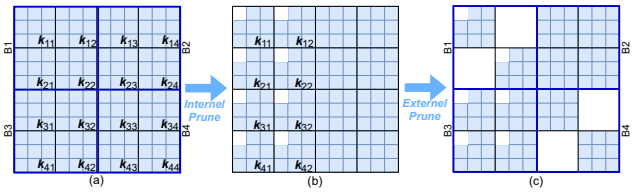

Figure 7: Joint Internal and External Pruning for Convolution Computation with $c_n = 2$.

the aforementioned diagonal kernels $\{\boldsymbol{K}_{21}, \boldsymbol{K}_{12}\}$ in Figure 6 actually form an *external structure* that, if pruned, contributes to eliminate one Perm operation over the convolved ciphertext containing two convolution as $\boldsymbol{K}_{21} * \boldsymbol{u}_1$ and $\boldsymbol{K}_{12} * \boldsymbol{u}_2$.

Putting things together: We define the *internal* and *external structure* and propose Hunter's HE-friendly structured pruning strategy for convolution, including *internal pruning* and *external pruning*, to achieve efficient convolution computation. Without accuracy loss, the former is to minimize the Perm operations needed to rotate each input ciphertext, while the latter is to minimize the Perm operations needed for rotating the (intermediate) convolved ciphertext to finally get the convolution output. Meanwhile, Hunter features with a joint and progressive pruning for kernels through firstly pruning the *internal structures* and then pruning the *external structures*. This gradual pruning strategy enables Hunter to smoothly reduce the model complexity and adaptively maintain the model performance for efficient privacy-preserving convolution without accuracy loss.

Figure 7 visually shows how Hunter combines *internal pruning* and *external pruning* to prune the kernel matrix with four filters, each of which has four kernels (in Figure 7 (a)), for a convolution layer. Here we set $c_n = 2$ and thus the four input channels are respectively encrypted into two input ciphertext. We first conduct *internal pruning* to prune an internal structure associated with the first input ciphertext (see Figure 7 (b)), over which one Perm operation is eliminated. Then we conduct *external pruning* to prune two external structures (see Figure 7 (c)), eliminating two Perm operations that would otherwise be conducted over the two convolved ciphertext respectively associated with the two input ciphertext.

3.3 Security Analysis

Similar to GAZELLE [18], the security of Hunter lies in the semantic security of packed HE algorithm, e.g., BFV [9], for linear computation, i.e., the convolution and dot product. Specifically, Hunter first identifies the Perm-intensive structures in the HE-based linear computation over the baseline models, e.g., AlexNet [20] and VGG [40], and accordingly prunes the original model(s) to minimize the Perm operations in the resultant network structure(s). The finally pruned models are used by the server for the MLaaS service, i.e., privacy-preserving prediction over the encrypted input from the client. Therefore, Hunter does not introduce any extra computation modules but a more efficient HE-based computation with lower complexity during the MLaaS process, compared with other frameworks such as GAZELLE. On the other hand, the security of nonlinear computation follows the same paradigm as GAZELLE, i.e., Garbled Circuits (GC) based ReLU computation, since Hunter targets at optimizing the linear computation only, which is orthogonal to the nonlinear counterpart.

4 EVALUATION

In this section, we evaluate Hunter's performance and compare it with GAZELLE [18]. Note that Hunter's pruning strategies are also applicable to other privacy-preserving deep learning frameworks based on packed HE such as DELPHI [26] and CrypTFlow2 [33]. We implement Hunter with Pytorch on a machine with Nvidia RTX5000 16G GPU, Intel Xeon Silver 4214 12-core CPU and 32G RAM. We conduct the experiments on six modern models, i.e., LeNet, AlexNet, VGG-11, VGG-13, VGG-16, and ResNet-32, with three mainstream datasets, i.e., MNIST [22], CIFAR-10 [19], and ImageNet [37]. The performance evaluation focuses on the computation cost. It is worth pointing out that Hunter maintains the same communication cost as the baseline privacy-preserving framework since the size of ciphertext exchanged between the server and client is the same.

Performance on Modern Deep Models. Table 1 compares Hunter and plaintext pruning, demonstrating that even with a higher pruning ratio, the plaintext pruning offers little help to decrease the Perm operations. In contrast, Hunter effectively reduces the HE computation (resulting in a low percentage of remaining Perm).

As shown in Table 2, the time to prune the models ranges from 1.2 hours (for LeNet on MNIST) to 17 hours (for VGG16 on ImageNet), which is affordable for the servers to complete offline pruning to obtain a ready-to-use model for privacy-preserving MLaaS. Without accuracy loss, Hunter efficiently prunes the original networks to reduce Perm, Mult, and Add operations. For example, in the state-of-the-art VGG-16 on ImageNet with 10 randomly chosen classes (see Table 7), the total number of Perm is reduced to as low as 2% of the original network, and at the same time, Mult and Add are reduced to only 14%.

The effectiveness of Hunter stems from its accurate, structured pruning strategy to eliminate Perm and Multi/Add operations. The experimental results show that larger networks have greater pruning ratio. This is because, for given task and input data, the larger networks often contain more redundancy in their kernels and weights that can be pruned without reducing the model accuracy. We also observe that *external pruning* contributes more to reducing Perm compared with *internal pruning*, because there are significantly more Perm involved in the external structure. In general, Hunter achieves higher performance gain in deeper models with large-scale datasets, e.g., as demonstrated in VGG-16 on ImageNet.

Visualization of Pruned Structures. In order to gain insights into Hunter's pruning strategies, Figures 8 and 9 respectively demonstrate the visualization of the pruning for one convolutional layer and one fully-connected layer in the LeNet with MNIST. Similar results are observed under other deep learning models.

As for pruning the convolutional layer, internal pruning features with column-wise pruning patterns (see Figure 8 (b) and the top part in (d)) since each rotated input ciphertext is multiplied with the internal structure, each of which contains the kernel elements from the same locations of the corresponding kernel (see Section 3.2). After the external pruning, specific kernel-packs are pruned such that the kernels in those kernel-packs are completely removed (see Figure 8 (c) and the bottom part in (d)). This resulted pattern is in accordance with our demonstration in Figure 7. Pruning the weight matrix of the fully-connected layer results in a weight matrix with

Table 1: Comparison between Hunter and plaintext pruning. Both schemes result in no accuracy loss. Each data entry shows the percent of remaining parameters and Perm operations after pruning. The plaintext pruning offers little help to reduce the HE-based Perm computation (as shown by high percentage of remaining Perm) even with deep pruning (i.e., low percentage of remaining parameters). Hunter effectively reduces the HE computation (with low percentage of remaining Perm).

Dataset	MNIST		CIFAR-10										ImageNet (10 classes)			
Models	LeNet		ResNet-32		AlexNet		VGG-11		VGG-13		VGG-16		AlexNet		VGG-16	
Pruning Schemes: Hunter (H) vs Plaintext (P) Pruning	H	P	H	P	H	P	H	P	H	P	H	P	H	P	H	P
Remaining Parameters (%)	39	9	52	46	10	10	9	9	11	10	10	9	32	10	3	5
Remaining Perm (%)	45	99	51	95	6	67	19	90	21	89	14	84	36	93	2	91

Table 2: The Hunter's computation performance compared with GAZELLE on six modern networks with three datasets. The pruned models maintain similar accuracy compared with the baseline models. Each data entry for computation performance shows the fraction or percentage to which the number of operations is reduced to, in comparison with the baseline model. For example, 2397/4312 (56%) means that the pruned model reduces the number of Perm from the original 4312 to 2397 (i.e., 56% of the original computation cost). Perm(in), Perm(ex), Perm(diag) are Perm involved in *internal*, *external*, *diagonal structures*.

	Dataset	MNIST	CIFAR-10					ImageNet (10 classes)	
	Models	LeNet	ResNet-32	AlexNet	VGG-11	VGG-13	VGG-16	AlexNet	VGG-16
	Parameters	44K	484K	21.6M	28.1M	28.3M	33.6M	58M	134M
	Model Accuracy								
	Baseline Accuracy (%)	99.34	92.25	77.48	92.4	94.18	93.91	78.8	93.8
	Hunter Pruned Accuracy (%)	99.37	92.2	78.88	92.66	94.09	94.06	79	94
	Computation Cost in Convolution and Dot Product								
Convolution	# Perm(in)	67/96 (70%)	2397/4312 (56%)	1498/5608 (27%)	2431/8984 (27%)	2789/9752 (29%)	3059/14872 (21%)	2952/5608 (53%)	2868/14872 (19%)
Convolution	# Perm(ex)	14/24 (58%)	6393/12800 (50%)	4125/92K (4%)	49K/256K (19%)	55K/261K (21%)	58K/409K (14%)	33K/92K (36%)	5003/409K (1%)
Convolution	# Mult	788/1350 (58%)	119.6K/231K (52%)	355K/1.9M (19%)	902K/4.6M (20%)	974K/4.7M (21%)	1.2M/7.4M (16%)	789K/1.9M (42%)	1M/7.4M (14%)
Convolution	# Add	774/1336 (58%)	119K/230K (52%)	355K/1.9M (19%)	901K/4.6M (20%)	972K/4.7M (21%)	1.2M/7.4M (16%)	788K/1.9M (42%)	1M/7.4M (14%)
Dot Product	# Perm(diag)	94/273 (34%)	23/23 (100%)	388/4373 (9%)	234/4629 (5%)	279/4629 (6%)	253/4629 (5%)	5143/16403 (31%)	433/33K (1%)
Dot Product	# Mult	93/272 (34%)	16/16 (100%)	952/8280 (12%)	593/8208 (7%)	638/8208 (8%)	430/8208 (5%)	5140/16K (31%)	434/33K (1%)
Dot Product	# Add	94/273 (34%)	23/23 (100%)	942/8198 (11%)	591/8206 (7%)	636/8206 (8%)	428/8206 (5%)	5143/16403 (31%)	433/33K (1%)
	Overall Model Computation Cost								
	# Perm	45%	51%	6%	19%	21%	14%	36%	2%
	# Mult	54%	52%	19%	20%	21%	16%	42%	14%
	# Add	54%	52%	19%	20%	21%	16%	42%	14%
	Model Pruning Time								
	Pruning Time (hours)	1.2	6.8	1.4	5.2	6.4	5.7	6.6	17

the elements arranged sparsely on diagonal lines (see Figure 9 (b)), and this pattern is consistent with Figure 4.

Layer-Wise Performance Breakdown. We further analyze the computation performance by breaking down a whole model (VGG-16 on CIFAR-10) into a stack of layers. The layer-wise performance breakdown for other deep models is given in Appendix A. In each layer, the computation complexity with respect to HE operations for linear functions, i.e., convolution and dot product, is compared between Hunter and GAZELLE. The detailed statistics are shown in Table 3 and Figure 10. The layer with a smaller number of kernels, e.g., Conv1, limits Hunter's pruning space, while the ones with greater number of kernels, e.g., Conv9 to Conv13, contain more redundancy for pruning. This trend is further visualized in Figure 10. We also observe that the increase of kernel dimension, i.e., the number of input and output channels, gives Hunter more pruning space, which results in an increased pruning ratio.

5 CONCLUSION

In this paper, we have proposed Hunter, which features an HE-friendly structured pruning scheme that, for the first time, efficiently prunes privacy-preserving deep models. Based on three novel HE-friendly structures, i.e., *internal structure*, *external structure*, and *weight diagonal*, Hunter outputs a pruned model that, without any loss in model accuracy, achieves a significant reduction in HE operations. We have implemented Hunter in various deep learning

Table 3: Layer-wise breakdown comparison between Hunter and GAZELLE (VGG-16 on CIFAR-10). Conv# denotes each convolution layer while Fc# represents each fully-connected layer of VGG-16. Each data entry for computation performance shows the fraction or percentage to which the number of operations is reduced to, in comparison with the baseline model. For example, 90/256 (35%) means that the pruned model reduces the number of Perm from the original 256 to only 90 (i.e., 35% of the original computation cost). Perm(in), Perm(ex), and Perm(diag) are Perm in *internal*, *external*, and *diagonal pruning*.

Conv Index	# Weights	# Perm(in)	# Perm(ex)	# Mult	# Add
Conv1	1.7K	24/24(100%)	-	1728/1728(100%)	1664/1664(100%)
Conv2	37K	90/256(35%)	252/1024(25%)	5984/18432(32%)	5952/18400(32%)
Conv3	74K	128/256(50%)	833/2048(41%)	15269/36864(41%)	15205/36800(41%)
Conv4	147K	307/512(60%)	2001/4096(49%)	38884/73728(53%)	38820/73664(53%)
Conv5	295K	256/512(50%)	3135/8192(38%)	61191/147456(41%)	61063/147328(41%)
Conv6	590K	512/1024(50%)	5951/16384(36%)	121567/294912(41%)	121439/294784(41%)
Conv7	590K	307/1024(30%)	3761/16384(23%)	74955/294912(25%)	74827/294784(25%)
Conv8	1.2M	205/1024(20%)	4430/32768(14%)	103257/589824(18%)	103001/589568(17%)
Conv9	2.4M	205/2048(10%)	7638/65536(12%)	138285/1179648(12%)	138029/1179392(12%)
Conv10	2.4M	410/2048(20%)	10599/65536(16%)	234259/1179648(20%)	234003/1179392(20%)
Conv11	2.4M	205/2048(10%)	7018/65536(11%)	140322/1179648(12%)	140066/1179392(12%)
Conv12	2.4M	205/2048(10%)	6756/65536(10%)	138712/1179648(12%)	138456/1179392(12%)
Conv13	2.4M	205/2048(10%)	6206/65536(9%)	139126/1179648(12%)	138870/1179392(12%)
Total	-	3059/14872(21%)	58580/408576(14%)	1213539/7356096(16%)	1211395/7353952(16%)
FC Index	# Weights	# Perm(diag)		# Mult	# Add
Fc1	2.1M	25/511(5%)		208/4096(5%)	200/4088(5%)
Fc2	16.8M	205/4095(5%)		206/4096(5%)	205/4095(5%)
Fc3	41K	23/23(100%)		16/16(100%)	23/23(100%)
Total	-	253/4629(5%)		430/8208(5%)	428/8206(5%)

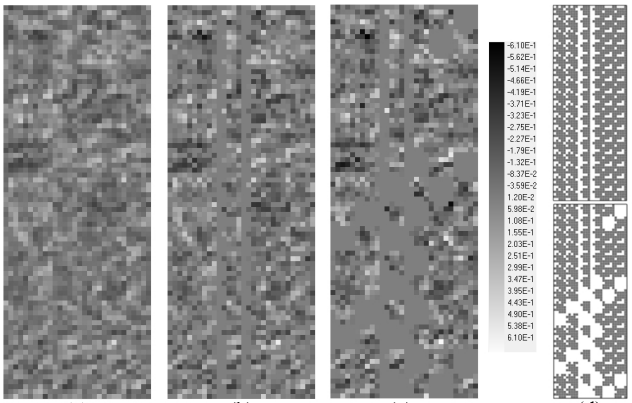

Figure 8: Visualization of pruning convolutional layer (LeNet on MNIST). (a) The original convolutional kernel matrix, (b) Kernel Matrix after internal pruning, (c) kernel matrix after external pruning, and (d) the binary representation of (b) at the top and (c) at the bottom where each pruned and unpruned value is in white and black, respectively.

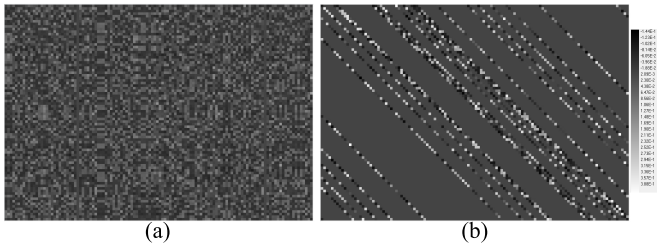

Figure 9: Visualization of pruning a weight matrix in LeNet with MNIST. (a) Original weight matrix, (b) Weight matrix after weight-diagonal pruning.

models, e.g., AlexNet, VGG and ResNet over classic datasets including MNIST, CIFAR-10 and ImageNet. The experiments have demonstrated that, without accuracy loss, Hunter efficiently prunes the original networks to reduce the HE Perm, Mult, and Add operations. For example, in the state-of-the-art VGG-16 on ImageNet with 10 chosen classes, the total number of Perm has been reduced to as low as 2% of the original network, and at the same time, Mult and Add have been reduced to only 14%, enabling a computation-efficient privacy-preserving MLaaS. Hunter represents the first step to effectively prune privacy-preserving deep learning models and may enlighten subsequent works to further close the performance gap in supporting privacy-preserving MLaaS.

ACKNOWLEDGMENTS

The authors would like to express special thanks of gratitude to Qiao as well as the lab members for helping and collaborating on this work and anonymous reviewers for the constructive comments. This work was supported in part by the National Science Foundation under Grants CNS-2120279, CNS-1950704, CNS-1828593, and OAC-1829771, Office of Naval Research under grant N00014-20-1-2065, National Security Agency under grants H98230-21-1-0165, H98230-21-1-0278, DoD Center of Excellence in AI and Machine Learning (CoE-AIML) under Contract Number W911NF-20-2-0277, and the Commonwealth Cyber Initiative.

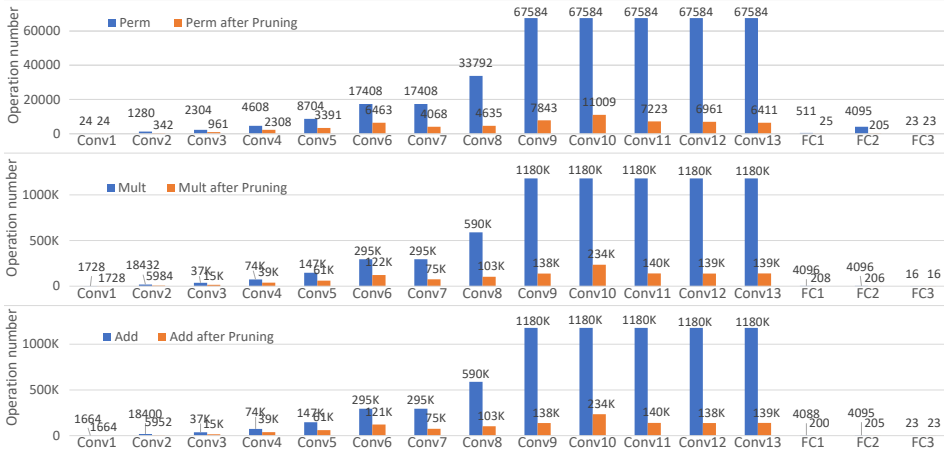

Figure 10: Visualization of layer-wise performance breakdown for VGG-16 on CIFAR-10.

REFERENCES

[1] Amazon.com. 2021. *Developer reference for Alexa interface.* https://developer.amazon.com/en-US/docs/alexa/device-apis/alexa-interface.html

[2] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *Proceedings of the ACM CCS*. 784–796.

[3] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. 2020. MP2ML: a mixed-protocol machine learning framework for private inference. In *Proceedings of the ARES*. 1–10.

[4] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*. Springer, 868–886.

[5] Zvika Brakerski, Craig Gentry, and Shai Halevi. 2013. Packed ciphertexts in LWE-based homomorphic encryption. In *Proceedings of the PKC*. 1–13.

[6] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. 1986. All-or-nothing disclosure of secrets. In *Proceedings of the EUROCRYPT*. 234–238.

[7] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Proceedings of the ASIACRYPT*. 409–437.

[8] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *Proceedings of the NDSS*.

[9] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.* (2012), 144.

[10] Zahra Ghodsi, Akshaj Veldanda, Brandon Reagen, and Siddharth Garg. 2020. Cryptonas: Private inference on a relu budget. *arXiv preprint arXiv:2006.08733* (2020).

[11] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the ICML*. 201–210.

[12] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. *arXiv preprint arXiv:1608.04493* (2016).

[13] Song Han. 2017. *Efficient methods and hardware for deep learning.* Ph. D. Dissertation. Stanford University.

[14] Song Han, Jeff Pool, John Tran, and William J Dally. 2015. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626* (2015).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE CVPR*. 770–778.

[16] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. 2021. DeepReDuce: Relu reduction for fast private inference. *arXiv preprint arXiv:2103.01396* (2021).

[17] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the ACM SIGSAC*. 1209–1222.

[18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *Proceedings of the USENIX Security*. 1651–1669.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Proceedings of the NeurIPS* 25 (2012), 1097–1105.

[21] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *Proceedings of the IEEE S&P*. 336–353.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[23] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the ACM SIGSAC*. 619–631.

[24] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE ICCV*. 2736–2744.

[25] Qian Lou, Song Bian, and Lei Jiang. 2020. Autoprivacy: Automated layer-wise parameter selection for secure neural network inference. *arXiv preprint arXiv:2006.04219* (2020).

[26] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *Proceedings of the USENIX Security*. 2505–2522.

[27] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the ACM SIGSAC*. 35–52.

[28] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *Proceedings of the IEEE S&P*. 19–38.

[29] Lucien KL Ng and Sherman SM Chow. 2021. GForce: GPU-Friendly Oblivious and Rapid Neural Network Inference. In *Proceedings of the USENIX Security*.

[30] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the EUROCRYPT*. 223–238.

[31] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved mixed-protocol secure two-party computation. In *Proceedings of the USENIX Security*.

[32] Arpita Patra and Ajith Suresh. 2020. BLAZE: blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042* (2020).

[33] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the ACM SIGSAC*. 325–342.

[34] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. {XONN}: Xnor-based oblivious deep neural network inference. In *Proceedings of the USENIX Security*. 1501–1518.

[35] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the ASIACCS*. 707–721.

[36] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the DAC*. 1–6.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[38] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE CVPR*. 815–823.

[39] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[40] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[41] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CRYPTGPU: Fast Privacy-Preserving Machine Learning on the GPU. *arXiv preprint arXiv:2104.10949* (2021).

[42] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 26–49.
[43] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229* (2020).
[44] Wei Wang, Sheng Wang, Jinyang Gao, Meihui Zhang, Gang Chen, Teck Khim Ng, and Beng Chin Ooi. 2018. Rafiki: Machine learning as an analytics service system. *arXiv preprint arXiv:1804.06087* (2018).
[45] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *Proceedings of the NeurIPS* 29 (2016), 2074–2082.
[46] Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V Phuong. 2018. GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning.. In *Proceedings of the IJCAI*. 3933–3939.
[47] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. 2021. GALA: Greedy ComputAtion for Linear Algebra in Privacy-Preserved Neural Networks. *arXiv preprint arXiv:2105.01827* (2021).

A LAYER-WISE BREAKDOWN OF COMPUTATION PERFORMANCE.

First give the complexity of DL Models on GAELLE framework in table 4. For the Conv layer, the input has c_i channels and c_n of them are pack encrypted in one ciphertext. The Conv layer has c_o filters and each filter contains c_i kernels. And the kernel size is $k_w k_h$. For FC layer, the size of input is n_i and n of them are pack encrypted in one ciphertext. The size of output is n_o. Thus the size of wights matrix of FC layer is $n_0 \times n_i$.

Table 4: Complexity of DL Models on GAELLE framework.

	#Perm(in)	#Perm(ex)	#Mult	#Add
Conv	$\frac{c_i(k_w k_h - 1)}{c_n}$	$\frac{c_i c_o (c_n - 1)}{c_n^2}$	$\frac{c_i c_o k_w k_h}{c_n}$	$\frac{c_o(c_i k_w k_h - 1)}{c_n}$
	# Perm(diag)		#Mult	#Add
FC	$\frac{n_i n_o}{n} - 1 + \log_2 \frac{n}{n_o}$		$\frac{n_i n_o}{n}$	$\frac{n_i n_o}{n} - 1 + \log_2 \frac{n}{n_o}$

The layer-wise breakdown of computation performance is shown below. Each data entry, e.g 17/24(71%), means prune operations from number of 24 to only 17, in other word, reserving 71% operations.

Table 5: Layer-Wise Performance Breakdown between Hunter and GAZELLE.

Layer Index	# Weights	# Perm	# Mult	# Add
LeNet with MNIST				
Conv1	150	17/24(71%)	108/150(72%)	102/144(71%)
Conv2	2.4K	64/96(67%)	680/1200(57%)	672/1192(56%)
Fc1	31K	57/128(45%)	57/128(45%)	57/128(45%)
Fc2	10K	19/127(15%)	20/128(16%)	19/127(15%)
Fc3	840	18/18(100%)	16/16	18/18
Total	44K	175/393(45%)	881/1622(54%)	868/1609(54%)
AlexNet with CIFAR-10				
Conv1	35K	360/360(100%)	34848/34848(100%)	34752/34752(100%)
Conv2	614K	4364/7296(60%)	193404/307200(63%)	193276/307072(63%)
Conv3	885K	344/25600(1%)	51277/442368(12%)	51085/442176(12%)
Conv4	1.3M	283/38400(1%)	42769/663552(6%)	42577/663360(6%)
Conv5	885K	272/26112(1%)	33151/442368(7%)	33023/442240(7%)
Fc1	1M	38/255(15%)	608/4096(15%)	592/4080(15%)
Fc2	17M	327/4095(8%)	328/4096(8%)	327/4095(8%)
Fc3	41K	23/23(100%)	16/16(100%)	23/23(100%)
Total	21.6M	6011/102141(6%)	356401/1898544(19%)	355655/1897798(19%)
VGG-11 with CIFAR-10				
Conv1	1.7K	24/24(100%)	1728/1728(100%)	1664/1664(100%)
Conv2	74K	2151/2304(93%)	35487/36864(96%)	35423/36800(96%)
Conv3	295K	3186/8704(37%)	67472/147456(46%)	67344/147328(46%)
Conv4	590K	4839/17408(28%)	112415/294912(38%)	112287/294784(38%)
Conv5	1M	6098/33792(18%)	130289/589824(22%)	130033/589568(22%)
Conv6	2M	11587/67584(17%)	210382/1179648(18%)	210126/1179392(18%)
Conv7	2M	16072/67584(24%)	207704/1179648(18%)	207448/1179392(18%)
Conv8	2M	7735/67584(11%)	136809/1179648(12%)	136553/1179392(12%)
Fc1	2M	51/511(10%)	416/4096(10%)	408/4088(10%)
Fc2	17M	160/4095(4%)	161/4096(4%)	160/4095(5%)
Fc3	41K	23/23(100%)	16/16(100%)	23/23(100%)
Total	28.1M	51926/269613(19%)	902879/4617936(20%)	901469/4616526(20%)

Table 6: Layer-Wise Performance Breakdown between Hunter and GAZELLE.

Layer Index	# Weights	# Perm	# Mult	# Add
VGG-13 with CIFAR-10				
Conv1	1.7K	24/24(100%)	1728/1728(100%)	1664/1664(100%)
Conv2	37K	2151/2304(93%)	17181/18432(93%)	17149/18400(93%)
Conv3	74K	256/256(100%)	33021/36864(90%)	32957/36800(90%)
Conv4	147K	307/512(60%)	39983/73728(54%)	39919/73664(54%)
Conv5	295K	307/512(60%)	75900/147456(51%)	75772/147328(51%)
Conv6	590K	512/1024(50%)	124748/294912(42%)	124620/294784(42%)
Conv7	1M	307/1024(30%)	143726/589824(24%)	143470/589568(24%)
Conv8	2M	410/2048(20%)	242296/1179648(21%)	242040/1179392(21%)
Conv9	2M	205/2048(10%)	141599/1179648(12%)	141343/1179392(12%)
Conv10	2M	205/2048(10%)	153732/1179648(13%)	153476/1179392(13%)
Fc1	2M	51/511(10%)	416/4096(10%)	408/4088(10%)
Fc2	17M	205/4095(5%)	206/4096(5%)	205/4095(5%)
Fc3	41K	15/15(100%)	16/16(100%)	23/23(100%)
Total	28.3M	3060/14373(21%)	974552/4710096(21%)	973046/4708590(21%)
ResNet-32 with CIFAR-10				
Conv1	432	24/24(100%)	432/432(100%)	416/416(100%)
Conv2	2.3K	63/128(49%)	553/1152(48%)	545/1144(48%)
Conv3	2.3K	46/128(36%)	360/1152(31%)	352/1144(31%)
Conv4	2.3K	55/128(43%)	454/1152(39%)	446/1144(39%)
Conv5	2.3K	63/128(49%)	508/1152(44%)	500/1144(44%)
Conv6	2.3K	82/128(64%)	661/1152(57%)	653/1144(57%)
Conv7	2.3K	64/128(50%)	476/1152(41%)	468/1144(41%)
Conv8	2.3K	23/128(18%)	155/1152(13%)	147/1144(13%)
Conv9	2.3K	29/128(23%)	257/1152(22%)	249/1144(22%)
Conv10	2.3K	52/128(41%)	427/1152(37%)	419/1144(37%)
Conv11	2.3K	38/128(30%)	360/1152(31%)	352/1144(31%)
Conv12	4.6K	168/192(88%)	1967/2304(85%)	1951/2288(85%)
Conv13	9.2K	291/384(76%)	3186/4608(69%)	3170/4592(69%)
Conv14	9.2K	195/384(51%)	2069/4608(45%)	2053/4592(45%)
Conv15	9.2K	197/384(51%)	1964/4608(43%)	1948/4592(42%)
Conv16	9.2K	272/384(71%)	3017/4608(65%)	3001/4592(65%)
Conv17	9.2K	261/384(68%)	2879/4608(62%)	2863/4592(62%)
Conv18	9.2K	207/384(54%)	2511/4608(54%)	2495/4592(54%)
Conv19	9.2K	103/384(27%)	1134/4608(25%)	1118/4592(24%)
Conv20	9.2K	307/384(80%)	3668/4608(80%)	3652/4592(80%)
Conv21	9.2K	225/384(59%)	2238/4608(49%)	2222/4592(48%)
Conv22	18.4K	582/640(91%)	8028/9216(87%)	7996/9184(87%)
Conv23	36.9K	709/1280(55%)	11188/18432(61%)	11156/18400(61%)
Conv24	36.9K	331/1280(26%)	5056/18432(27%)	5024/18400(27%)
Conv25	36.9K	368/1280(29%)	5434/18432(29%)	5402/18400(29%)
Conv26	36.9K	574/1280(45%)	8608/18432(47%)	8576/18400(47%)
Conv27	36.9K	601/1280(47%)	8517/18432(46%)	8485/18400(46%)
Conv28	36.9K	703/1280(55%)	10549/18432(57%)	10517/18400(57%)
Conv29	36.9K	747/1280(58%)	11810/18432(64%)	11778/18400(64%)
Conv30	36.9K	600/1280(47%)	9699/18432(53%)	9667/18400(53%)
Conv31	36.9K	810/1280(63%)	11416/18432(62%)	11384/18400(62%)
Fc1	23K	23/23(100%)	16/16(100%)	23/23(100%)
Total	484.3K	8813/17135(51%)	119597/230848(52%)	119028/230279(52%)

Table 7: Layer-Wise Performance Breakdown between Hunter and GAZELLE.

AlexNet with ImageNet (10 classes)				
Layer Index	# Weights	# Perm	# Mult	# Add
Conv1	35K	360/360(100%)	34848/34848(100%)	34752/34752(100%)
Conv2	614K	3376/7296(46%)	160160/307200(52%)	160032/307072(52%)
Conv3	885K	7573/25600(30%)	157856/442368(36%)	157664/442176(36%)
Conv4	1.3M	13042/38400(34%)	221514/663552(33%)	221322/663360(33%)
Conv5	885K	11981/26112(46%)	214767/442368(49%)	214639/442240(49%)
Fc1	37.7M	3687/12285(30%)	3690/12288(30%)	3687/12285(30%)
Fc2	17M	1433/4095(35%)	1434/4096(35%)	1433/4095(35%)
Fc3	41K	23/23(100%)	16/16(100%)	23/23(100%)
Total	58M	41475/114171(36%)	794285/1906736(42%)	793552/1906003(41%)
VGG-16 with ImageNet (10 classes)				
Conv1	1.7K	24/24(100%)	1728/1728(100%)	1664/1664(100%)
Conv2	37K	43/1280(3%)	1946/18432(11%)	1914/18400(10%)
Conv3	74K	77/2304(3%)	4251/36864(12%)	4187/36800(11%)
Conv4	147K	301/4608(7%)	14679/73728(20%)	14615/73664(20%)
Conv5	295K	696/8704(8%)	31078/147456(21%)	30950/147328(21%)
Conv6	590K	781/17408(4%)	58130/294912(20%)	58002/294784(20%)
Conv7	590K	733/17408(4%)	57880/294912(20%)	57752/294784(20%)
Conv8	1M	2183/33792(6%)	148367/589824(25%)	148111/589568(25%)
Conv9	2M	957/67584(1%)	172764/1179648(15%)	172508/1179392(15%)
Conv10	2M	877/67584(1%)	172514/1179648(15%)	172258/1179392(15%)
Conv11	2M	390/67584(1%)	123666/1179648(10%)	123410/1179392(10%)
Conv12	2M	351/67584(1%)	118467/1179648(10%)	118211/1179392(10%)
Conv13	2M	458/67584(1%)	118884/1179648(10%)	118628/1179392(10%)
Fc1	103M	287/28665(1%)	294/28672(2%)	287/28665(1%)
Fc2	17M	123/4095(3%)	124/4096(5%)	123/4095(3%)
Fc3	41K	23/23(100%)	16/16(100%)	23/23(100%)
Total	134M	8304/456231(2%)	1024788/7388880(14%)	1022643/7386735(14%)

∗The selected 10 classes are: n04552348-warplane, n03670208-limousine, n01560419-bulbul, n02123394-Persiancat, n02415577-bighorn, n02099601-goldenretriever, n01641577-bullfrog, n02389026-sorrel, n04147183-schooner, n04467665-trailertruck.