2024 Annual Computer Security Applications Conference (ACSAC) | 979-8-3315-2088-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ACSAC63791.2024.00096

TILE: Input Structure Optimization for Neural Networks to Accelerate Secure Inference

Yizhou Feng
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA, USA
yfeng002@odu.edu

Qiao Zhang
College of Computer Science
Chongqing University
Chongqing, China
qiaozhang@cqu.edu.cn

Yifei Cai
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA, USA
ycai001@odu.edu

Hongyi Wu
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ, USA
mhwu@arizona.edu

Chunsheng Xin*
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA, USA
cxin@odu.edu

***Abstract*—Machine Learning as a Service (MLaaS) is an innovative framework that enables a broad range of users to capitalize on the powerful Artificial Intelligence (AI) technologies. Nevertheless, MLaaS raises a privacy concern for both the client data and server model. To address this issue, several Secure Inference (SI) frameworks for MLaaS have been proposed in the literature that take advantage of Homomorphic Encryption (HE) operations. However, the computation cost of these frameworks is still high, especially for real-time applications. In this paper, we propose a novel system called *input structure optimization for neural networks* (TILE) to accelerate SI. The goal of TILE is to reduce both linear and non-linear computation costs, as well as non-linear communication costs in MLaaS, while maintaining the model accuracy. TILE defines two novel HE-friendly input structures: Internal Tile and External Tile Structures, aimed at reducing the HE operations for SI. We also develop a search mechanism to identify optimal application locations for these input structures. We apply TILE to widely used models such as VGG and ResNet, and datasets including Cifar10 and Tiny-ImageNet. The experimental results demonstrate that TILE effectively reduces the computation time, with up to 51.57% reduction for a state-of-the-art SI framework. Furthermore, TILE can also be applied to models that have already been pruned to significantly reduce the computation time, to further reduce the overall computation time by 25.90%.**

***Index Terms*—Input Structuring, Machine Learning as a Service, Privacy-preserving Computation, Packed Homomorphic Encryption**

* *Corresponding Author*

1. Introduction

Machine Learning as a Service (MLaaS) is a framework that enables a broad range of users to access powerful artificial intelligence (AI) models for numerous applications, such as image classification [1], [2], facial recognition [3], credit-risk assessment [4] and disease diagnosis for patients [5], [6], [7]. In the MLaaS setting, a client sends its data to a server, typically in a cloud, which runs its AI model and returns the inference result to the client. However, the privacy concern raises a great challenge, as the client data can be sensitive, such as medical records of patients. For example, several legislations prevent the exposure of certain private data, such as the Health Insurance Portability and Accountability Act (HIPAA) in the US, the General Data Protection Regulation (GDPR) in the EU, and the Personal Data Protection Act (PDPA) in Singapore. On the other hand, the server also has a strong need to safeguard its AI model, specifically the model parameters, to protect its intellectual property, since training AI models usually needs substantial investments in high-quality training data, hardware, and algorithmic design. Exposure of these parameters can result in adverse business competition implications.

To address the privacy concern of both client data and server models, *secure inference* (SI) has been developed, that keeps client data undisclosed to the server while concealing server model parameters from clients. Prior works on SI embed cryptographic primitives into computation processes of Deep Learning (DL) models [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]. Based on DL functionality, SI computations for an AI model can be divided into linear and non-linear parts. For linear computations such as convolution and dot product,

implementing them using Homomorphic Encryption (HE) [23], [24] is preferable, as it supports linear operations directly over the ciphertext domain. Non-linear computations, including ReLU, truncation, and max pooling, are usually implemented using cryptographic primitives such as Garbled Circuit (GC) [25], [26], Oblivious Transfer (OT) [27], [28], [29], and Secret Sharing (SS) [30]. Combining both linear and non-linear implementations ensures that the entire SI framework can replicate the functionality of the original DL model while ensuring efficient and privacy-preserved execution. Nevertheless, even with the state-of-the-art SI frameworks [15], [17], [13], [19], [18], the computational overhead remains far from practical. This issue becomes even more profound for deeper AI models with larger inputs. Additionally, another observation in [18] shows that the computation overhead of state-of-the-art SI frameworks, such as Gazelle [15], Gala [18], and CrypTFlow2 [13], is primarily dominated by HE-based linear operations and OT-based non-linear operations, respectively. The cost of these operations relies on two key components: the complexity of input data and the complexity of model parameters. Reducing both parameter complexity and input data complexity can enhance computational efficiency for SI. In the literature, model pruning is a commonly used method to reduce the parameter size of a given model. For example, frameworks such as Hunter [31] and MOSAIC [32] have been developed to conduct model pruning for SI. These frameworks construct HE-friendly structures for model parameters in SI frameworks, which effectively reduce the time required for SI. However, due to the limitations of model pruning strategies and SI framework settings, these solutions still incur high costs associated with Internal Perm operations and their corresponding computations. Additionally, there are no practical methodologies for reducing input data complexity in SI. Therefore, a new approach is needed to reduce input data complexity, which could enhance the efficiency of SI.

In this paper, we propose a system callsed *input structure optimization for neural networks* (TILE) to accelerate SI. Our work began with a crucial observation that ReLU layers in DL models often produce similar output values due to their inherent property of nullifying negative values. Consequently, if the ReLU layer's output values are similar, the input values are likely to be similar as well. By substituting these similar values with a single value, such as an average, we can reduce input data complexity and lower inference time for SI with minimal impact on final predictions. Another critical observation is that the later layers of a DL model have larger receptive fields that focus less on input details [33], [34], [35], [36]. Therefore, the slight approximation introduced by substituting multiple ReLU output values with a single value does not significantly impact the final predictions of a DL model. Motivated by these observations, our first objective is to design a HE-friendly input data structure to reduce subsequent HE-based computations within the convolution layers and the corresponding non-linear OT-based computations preceding these layers.

Secondly, the computational overhead in HE-based calculations primarily comes from the high computation complexity of three fundamental HE operations: Addition (Add), Multiplication (Mult), and Permutation (Perm). Our experiments showed that the Perm operation has the highest time complexity compared to other HE operations, a finding consistent with prior studies [31], [32], [18]. We have also observed that Perm operations, along with the associated Mult operations, predominantly contribute to the computational overhead of a single convolution operation. We refer to these Perm operations within this context as "Internal Perm". Consequently, our second objective focuses on minimizing the number of Internal Perm operations (as well as their associated Mult and Add operations) to reduce the computation cost of SI.

To the best of our knowledge, TILE is the first work that focuses on optimizing the input structure for SI computation. Moreover, it is complemental with the existing model compression techniques such as the model distillation [37] and the recent HE-friendly model pruning technology [32], and can further improve the performance of the compressed/pruned models. To this end, TILE provides *a new dimension* to improve SI frameworks and sheds lights to further close the gap toward practical SI.

In summary, our contributions are as follows.

- We propose a system called *input structure optimization for neural networks* (TILE) [1] for SI. TILE employs two efficient structures, Internal Tile and External Tile, designed to be HE-friendly. These structures effectively reduce the HE operations required for linear computation and the OT operations required for corresponding non-linear computation, thereby reducing the overall cost for SI.
- We construct a search mechanism for TILE, to identify optimal locations within a model to apply these tile structures. Moreover, model reconstruction is performed based on the search results obtained. This refined model maintains the same level of accuracy as the original model while reducing the computation time needed for SI.
- The experiments demonstrate that TILE efficiently speeds up SI. TILE achieves a reduction of up to 45.81% in Internal Perm, 54.64% in Mult and Add operations, and 32.29% corresponding ReLU operations, resulting in a 41.20% reduction in SI time for state-of-the-art MLaaS frameworks. Even for a pruned AI model that has already had more than 57.62% time reduction, TILE achieves an additional reduction of up to 25.90% in SI time for state-of-the-art frameworks.

The rest of the paper is organized as follows. In Section 2 we discuss the system model, cryptography tool, and threat model adopted in our system. Section 3 elaborates on the proposed system and gives the security and complexity

1. Our implementation is available at https://github.com/yizhouf743/TILE

analysis. Section 4 presents the performance evaluation. Finally, we conclude the paper and discuss the future work in Section 5.

2. Preliminaries

2.1. System Model

Our work focuses on deep Convolution Neural Networks (CNNs), where the convolution operation predominantly determines the linear computation cost. The computation of convolution involves placing the kernel at each location of the input data, followed by summing up the element-wise products between the kernel values and the corresponding input feature map data within the kernel window. Non-linear computations in DL models usually employ the ReLU activation function in modern CNNs such as VGG[34] and ResNet[38], represented as $f(x) = \max\{0, x\}$. Additionally, the pooling operation, typically applied after convolution, downsamples the output data. Our work aims to optimize the SI computation efficiency of convolution operations that accordingly reduces the associated non-linear operations. It can be adopted in state-of-the-art SI frameworks such as Gazelle[15] and CrypTFlow2[13], to speed up their inference.

2.2. Thread Model

We assume a semi-honest threat model that is widely adopted in SI frameworks [39], [15], [17], [13], [18]. Specifically, the two parties in TILE are non-colluding and follow the protocols but may try to infer the other party's data from their received messages. For instance, the client may try to infer the model parameter, such as values of kernels and weight matrix. The server may try to figure out the client's private input. We perform security analysis for TILE in Section 3.4.

2.3. Packed Homomorphic Encryption

Following state-of-the-art Secure Inference (SI) frameworks [15], [13], [17], [18], our work utilizes the Packed Homomorphic Encryption (PHE) scheme [24] rather than the Fully Homomorphic Encryption (FHE) [40] scheme. While FHE offers the advantage of supporting an unlimited number of homomorphic operations on encrypted data through bootstrapping, PHE has a great advantage on the computational efficiency, thanks to its Single Instruction Multiple Data (SIMD) method that enables parallel processing of multiple data points within a single ciphertext. Our work adopts the BFV scheme implementation for PHE, that supports homomorphic Addition (Add, $\oplus$), Multiplication (Mult, $\otimes$), and Permutation (Perm) operations. For example, let x and y be two plaintext vectors with m values each, encrypted into two ciphertexts by the client using PHE as $[x]_c$ and $[y]_c$, respectively. The Add operation outputs ciphertext $[x+y]_c$ by element-wise addition of $[x]_c$ with $[y]_c$. The Mult operation outputs ciphertext $[x \times y]_c$ by element-wise multiplication between $[x]_c$ and y. The Perm operation cyclically rotates the values of a ciphertext, such that rotating $[x]_c$ results in another ciphertext where the value at the i-th position moves to the first position. Meanwhile, multiple Perm operations on the same ciphertext can be reduced by performing one Perm Decomposition operation with the same number of Hoisted Perm operations[15], thus amortizing the total Perm time. It is worth noting that many SI frameworks, such as [17], [15], [13], [18], primarily focus on reducing the number of Perm operations (External Perm) required to align the intermediate ciphertext. This approach enables the client to obtain the convolution output in the same data order as it would from the convolution operation in plaintext while incurring minimal SI cost.

However, a significant computational cost remains associated with ciphertext Mult and Internal Perm operations in convolution computations. These operations significantly dominate the time required for SI. For example, when applying the CrypTFlow2 framework [13] to VGG16 with Tiny-ImageNet, we find that Mult and Perm operations together constitute up to 90.20% of the total computation time for a single convolution layer. To further enhance the efficiency of SI, TILE aims to minimize the computational complexity of the Internal Perm operation and its associated Mult and Add operations, thereby reducing the cost of SI.

3. System Overview of TILE

Central to TILE are two HE-friendly input data structures: Internal Tile and External Tile, which shape input feature maps to effectively reduce three HE operations (Mult, Add, and Internal Perm) within convolution computations and minimize the associated non-linear operations, while preserving the model's accuracy. Next, we describe the Internal and External Tile structures and how they are applied in the HE-based convolution computations.

3.1. Tile Structure for Convolution Computation

For a convolution layer, we denote the input data u has C_i channels, each with a size of $u_h \times u_w$, while the plaintext kernel k has C_o channels, each with a size of $C_i \times k_h \times k_w$. Each ciphertext can store C_n channels of the input data u. The client encrypts input u as $[u]_c$, which is then sent to the server. The server conducts convolution computation between $[u]_c$ and its kernel k to obtain the encrypted output $[v]_c$, where $v = k * u$ and "$*$" represent the convolution operation.

3.1.1. Single Input and Single Output (SISO) Convolution. In SI frameworks such as [15], [13], when $C_i = C_o = 1$, it is called *single input and single output* (SISO) convolution. The SISO convolution computation proceeds by initially placing the kernel at each location within the input. Subsequently, the process involves the summation of element-wise products between the values of the kernel and the corresponding elements of the input data within the

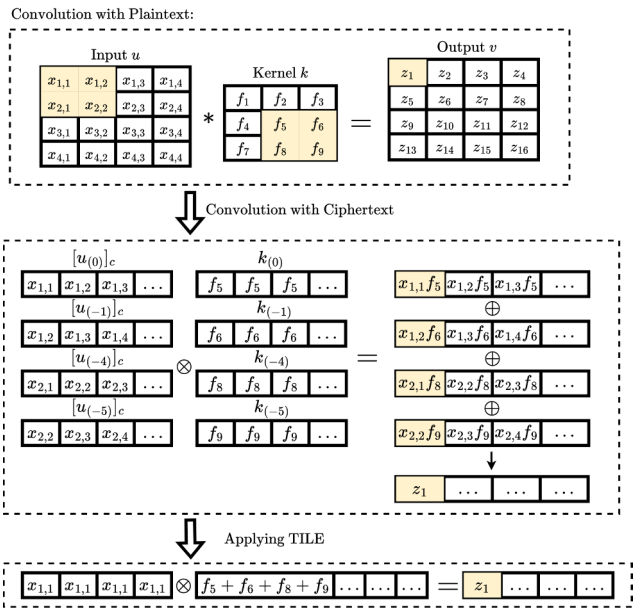


Figure 1: Toy example for computing the first convolution output value in SISO

predefined kernel window. As a result, the i-th value of the convolution output can be formally described as the inner product of vectors constructed from the kernel centered around the i-th position within the input.

Next we use a toy example in Figure 1, where $C_i = C_o = 1$, $C_n = 1$, $u_h = u_w = 4$, and $k_h = k_w = 3$, to illustrate HE convolution computation and how the Internal Tile structure can reduce the computation time. In plaintext, the first value of the convolution output, denoted as z_1, is computed as follows:

$$z_1 = x_{1,1}f_5 + x_{1,2}f_6 + x_{2,1}f_8 + x_{2,2}f_9. \quad (1)$$

However, in SI frameworks based on PHE, the server's computation follows a different process. The server computes: $[v]_c = \sum_{i\in L}[u_{(i)}]_c \otimes k_{(i)}$, where $L = \{0, -1, -4, -5\}$ represents the list of rotation steps. The server performs multiplication between the plaintext kernels $k_{(0)}$, $k_{(-1)}$, $k_{(-4)}$, and $k_{(-5)}$ and the encrypted input $[u_{(0)}]c$, along with its rotated ciphertexts $[u(-1)]c$, $[u(-4)]c$, and $[u(-5)]_c$. These results are then summed to produce the encrypted convolution output $[v]_c$, with its first element representing the encrypted value of z_1, as illustrated in Figure 1.

An interesting observation is that the number of Perm, Mult, and Add operations exhibits a linear relationship with the number of terms in (1). Specifically, to obtain each term in (1), it requires one Internal Perm (except for terms that include the kernel value at the center of the kernel, such as f_5 in Figure 1), and a Mult operation, followed by an Add operation (except the last term). For this example, it needs total *three Perm, four Mult, and three Add operations*.

However, if the values within the input feature map are similar, i.e., $x_{1,1} \approx x_{1,2} \approx x_{2,1} \approx x_{2,2}$, then (1) can be rewritten as follows.

$$z_1 = x_{1,1}(f_5 + f_6 + f_8 + f_9)$$

As a result, the Internal Perm, Mult, and Add operations each is reduced by three. That is, it would need only one Mult operation (Note that $f_5 + f_6 + f_8 + f_9$ is done at the plaintext domain).

Motivated by those observations, we partition the entire input feature map into fixed-size structural units, called Internal Tiles, with each unit representing an PxP block of data. For example, Figure 3 illustrates the resulting input feature map with 2x2 partitioning, denoted as u'. Within each unit/tile, if the values are similar, then all values are substituted with their corresponding mean value. We use different colors to distinguish different units. For example, in u', $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}$ form a 2x2 tile, represented by the yellow color, and they have the same value (the original mean). Note that the individual values in a unit change their locations after a ciphertext rotation. But the values in the same unit can be still seen by their color.

Given that all pixels in a tile have the same value, the Mult operations for rotated ciphertexts $[u_{(+4)}]c, [u_{(+1)}]c, [u_{(-1)}]c$, and $[u_{(-4)}]c$ in Figure 2 can be absorbed into the Mult operations of other ciphertexts, as illustrated in Figure 3. For example, consider the computation of the sixth convolution output, denoted as z_6:

$$z_6 = x_{1,1}f_1 + x_{1,2}f_2 + x_{1,3}f_3 + x_{2,1}f_4 + x_{2,2}f_5 + x_{2,3}f_6 + x_{3,1}f_7 + x_{3,2}f_8 + x_{3,3}f_9.$$

In SI frameworks, the server needs to compute $v = \sum_{i\in L}[u_{(i)}]_c \otimes k_{(i)}$ for the rotation steps in the list $L = \{-5, -4, -3, -1, 0, +1, +3, +4, +5\}$ to obtain z_6. This computation involves 8 Internal Perm, 9 Mult and 8 Add operations.

As illustrated in Figure 3, since in u', $x_{1,1} = x_{1,2} = x_{2,1} = x_{2,2}$, $x_{1,3} = x_{2,3}$, and $x_{3,1} = x_{3,2}$, z_6 can be rewritten as follow.

$$\begin{aligned} z_6 &= (x_{1,1}f_1 + x_{1,2}f_2 + x_{2,1}f_4 + x_{2,2}f_5) + (x_{1,3}f_3 + x_{2,3}f_6) \\ &\quad + (x_{3,1}f_7 + x_{3,2}f_8) + x_{3,3}f_9 \\ &= x_{2,2}(f_1 + f_2 + f_4 + f_5) + x_{1,3}(f_3 + f_6) + x_{3,1}(f_7 + f_8) \\ &\quad + x_{3,3}f_9 \end{aligned}$$

In other words, the computation of terms $x_{1,1}f_1, x_{1,2}f_2, x_{2,1}f_4, x_{2,3}f_6$ and $x_{3,2}f_8$ are absorbed into the computation of other terms in the expression. As a result, the HE operations corresponding to ciphertext $[u_{(+3)}]c$, $[u_{(+1)}]c, [u_{(-1)}]c$, and $[u_{(-3)}]_c$ are eliminated, including 4 operations each for Perm, Mult, and Add. Now, the server can obtain z_6 by computing $v = \sum_{i\in L'}[u'_{(i)}]_c \otimes k'_{(i)}$ for the rotation steps i in the list $L' = \{-5, -3, 0, +3, +5\}$. This computation only involves 4 Internal Perm, 5 Mult, and 4 Add operations. Recall that u represents the input of the convolution layer as well as the output of the preceding ReLU layer, denoted as $u = f(Y)$. Here, $Y = y_{i,j}$ $(0 \le i \le u_h, 0 \le j \le u_w)$ is the input of the ReLU layer. If $x_{1,1} \approx x_{1,2} \approx x_{2,1} \approx x_{2,2}$, the input values to

the preceding ReLU layer are likely similar as well, i.e., $y_{1,1} \approx y_{1,2} \approx y_{2,1} \approx y_{2,2}$. This makes it possible to eliminate non-linear computations for $f(y_{1,2})$, $f(y_{2,1})$, and $f(y_{2,2})$. Similarly, leveraging $x_{1,3} \approx x_{2,3}$ and $x_{3,1} \approx x_{3,2}$, the computation of $f(y_{2,3})$ and $f(y_{3,2})$ can eliminated. Similar observations can also be found for other non-linear computations, such as truncation and max pooling.

In the implementation, after applying the Internal Tile structuring of input data, the client transmits the reshaped data, reduced to $\frac{1}{P}$ the size of the original data, to the server for OT-based ReLU computation, since only one pixel value from each tile is needed (other pixel values in a tile are the same). The client share of the output from this ReLU computation, which serves as the input for the subsequent HE-based convolution computation, is also sent, after encryption, to the server for further processing. To ensure that the ReLU output size matches the required size for the subsequent convolution layer input, each pixel in the ReLU output is expanded into a block of (P, P) pixels, each with the same value, based on the predefined tile positions. The Internal Tile structuring reduces not only the computation cost including HE operations and ReLU, but also the communication cost, thereby accelerating the SI computation. It is noteworthy that applying the Internal Tile structure is different from the mean pooling operation in CNN, as the Internal Tile structure retains the original spatial structure while reducing the required HE operations, and the substitution using a single value within each tile unit preserves relationships between neighboring pixels, enabling it to capture structural connections and prevent information loss.

3.1.2. Multiple Input Multiple Output (MIMO) Convolution. Next we consider the general case of MIMO convolution, where the number of input channels C_i and/or the number of output channels C_o are greater than one. In MIMO computation, we have C_i input channels (each with a size of $u_w \times u_h$) convolved with C_o filters (each with a size of $C_i \times f_w \times f_h$) to produce C_o output channels (each with the size of $u_w \times u_h$). Under the state-of-the-art SI framework introduced in [13], which is based on the Grouped Out-Rot MIMO method, let us consider a scenario with the following setting: $C_i = 8$, $C_o = 4$, and $C_n = 4$, where C_n represents the number of input channels packed into one ciphertext. As depicted in Figure 4, the client encodes eight input channels, denoted as H_j $(1 \le j \le 8)$, into two ciphertexts: $[u_1]_c, [u_2]c$. These ciphertexts are then convolved with filters $K = \{k_{i,j}\}$ $(1 \le i \le 4, 1 \le j \le 8)$ to generate an encrypted convolution output, $[V]_c$, comprising four output channels: v_1, v_2, v_3, v_4.

For example, the computation of the first output channel, denoted as v_1, is as follows:

$$v_1 = H_1 * k_{1,1} + H_2 * k_{1,2} + H_3 * k_{1,3} + H_4 * k_{1,4} + H_5 * k_{1,5} + H_6 * k_{1,6} + H_7 * k_{1,7} + H_8 * k_{1,8}.$$

Here, each term is an individual convolution operation between one input channel and one kernel, using the SISO method above.

In this example, excluding the computational cost within individual convolution operations, the server computes

$$[V]_c = \sum_{j=1}^{4} Perm(\sum_{i=1}^{2}([u_i]_c \otimes B_{i,j}), j-1).$$

This process involves 8 Mult operations for the ciphertexts $[u_1]_c$ and $[u_2]_c$, each paired with a corresponding kernel vector $B_{i,j}$, along with 7 Add to combine their results, as illustrated in Figure 4. Additionally, 3 Perm operations are needed to ensure the elements in the three ciphertexts resulting from the Mult operations are in the right order to be added to obtain the convolution output, $\{v_i\}$ $(1 \le i \le 4)$. Here each unit to be permuted is the result of an SISO convolution operation and hence the Perm operation is called External Perm.

One observation is that if most values in a group of input channels are similar, we can reduce the computation time. For example, if the H_5 is similar to H_6 and H_7 is similar to H_8 in $[u_2]_c$, i.e., $H_5 \approx H_6$ and $H_7 \approx H_8$, the convolution computation for v_1 can be rewritten as follows:

$$v_1 = H_1 * k_{1,1} + H_2 * k_{1,2} + H_3 * k_{1,3} + H_4 * k_{1,4} + H_5 * (k_{1,5} + k_{1,6}) + H_7 * (k_{1,7} + k_{1,8}).$$

Similarly, in the SI framework, the computation for $[V]_c$ can be simplified as follows.

$$[V]_c = \sum_{j=1}^{4} \begin{cases} Perm([u_1]_c \otimes B_{1,j} \oplus [u_2']_c \otimes B_{2,j}', j-1), & \text{if } j = 1 \text{ or } 3 \\ Perm([u_1]_c \otimes B_{1,j}, j-1), & \text{otherwise} \end{cases}$$

This optimization reduces the computation to 6 Mult and 5 Add operations, as depicted in Figure 5. Moreover, similar to our discussions with the Internal Tile structure, the required number of OT-based ReLU computations for the preceding ReLU layer can also be reduced by 25% given that $H_5 \approx H_6$ and $H_7 \approx H_8$. As a result, the data size that the client needs to transmit to the server is also reduced by 25%.

These group structural units within the input feature map in MIMO convolution are referred to as "External Tile". This structure effectively minimizes the computational cost of 1×1 Convolution layers. To apply this structure to the HE-based convolution computation, the client initially partitions the input channels into External Tile structures, each containing $M(0 < M \le \frac{C_n}{2})$ input channels. If for a group of input channels, all feature pixels at the same position among those input channels are similar, then those input channels form an External Tile. Within each tile, the input channels conduct element-wise averaging, i.e., for each input channel, the value of a feature pixel is substituted by the average value of the feature pixels at the same position among all input channels in a tile. The External Tile structure reduces the required Mult and Add operations. Moreover, the computation and communication costs in the preceding OT-based ReLU layer associated with the redundant channels in an External Tile can be eliminated, similarly as in the case of the Internal Tile structure.

The client transmits the reshaped data, reduced to $\frac{1}{M}$

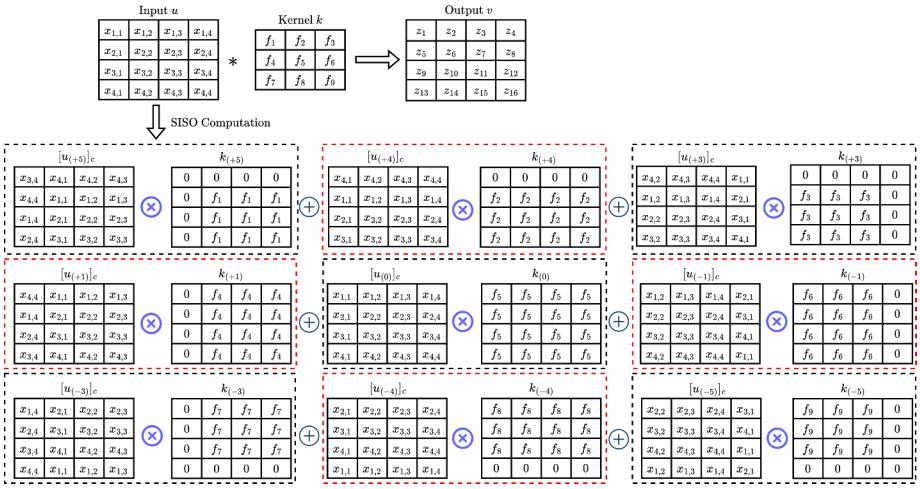


Figure 2: SISO convolution in [15], [13]. The red dashed windows indicate the operations to be eliminated by TILE in Figure 3

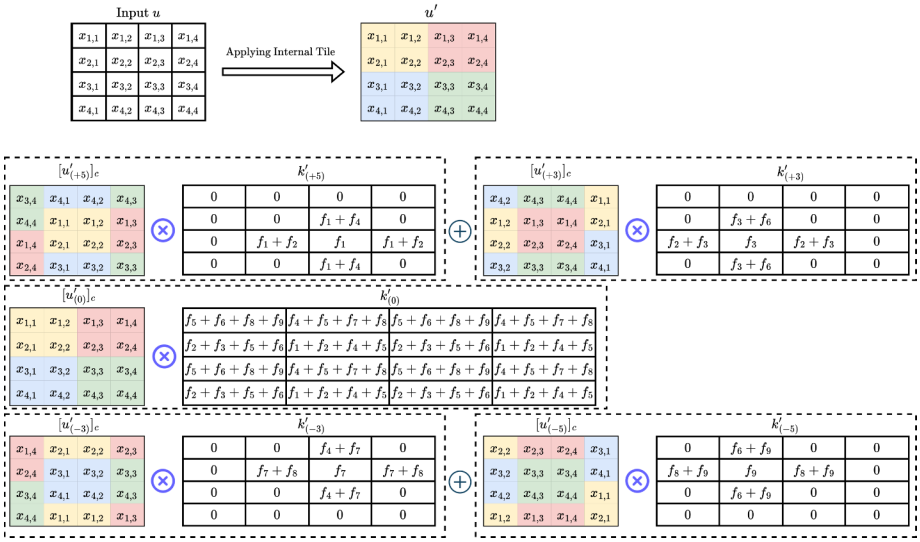


Figure 3: SISO convolution applied with the Internal Tile structure (Pixels with the same color belong to the same tile)

of the size of the original data, to the server for OT-based ReLU computation, since only one input channel's data from each tile is required (the data of other channels in a tile are the same). Then, the client share of the ReLU output, after encryption, is sent to the server for further computation. Since the output of ReLU computation usually serves as the convolution input feature map of the subsequent layer, the client replicates each output channel of the ReLU output by a factor of M based on predefined tile positions, to match the required input size for the subsequent convolution. This entire process enhances feature sharing, optimizes the SI efficiency, and retains essential information without compromising the correctness of the computation.

3.2. Tile Position Locator

In practice, blindly applying the aforementioned tile structure to entire input feature maps can lead to the loss of edge information and decrease model accuracy. The extent of the loss of accuracy varies depending on the applied position and the model architecture. In deep neural networks, convolution layers closer to the final output often capture more abstract and high-level features, as pointed out by multiple studies [33], [34], [35], [36]. These deep layers have larger receptive fields and are less sensitive to input details, which makes them more tolerant to information loss without significantly affecting model behavior and accuracy. In the context of pruned models, the sensitivity to input details is also influenced by the number of input channels, as discussed in [41]. The reduction of input channels leads to decreased redundancy in the input data, which poses a

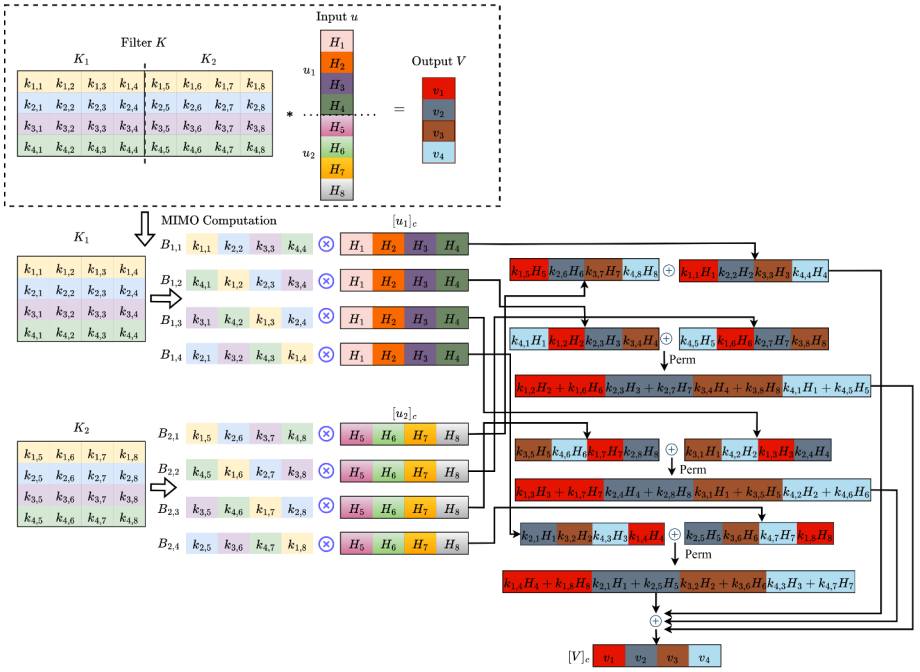


Figure 4: MIMO convolution in [13]

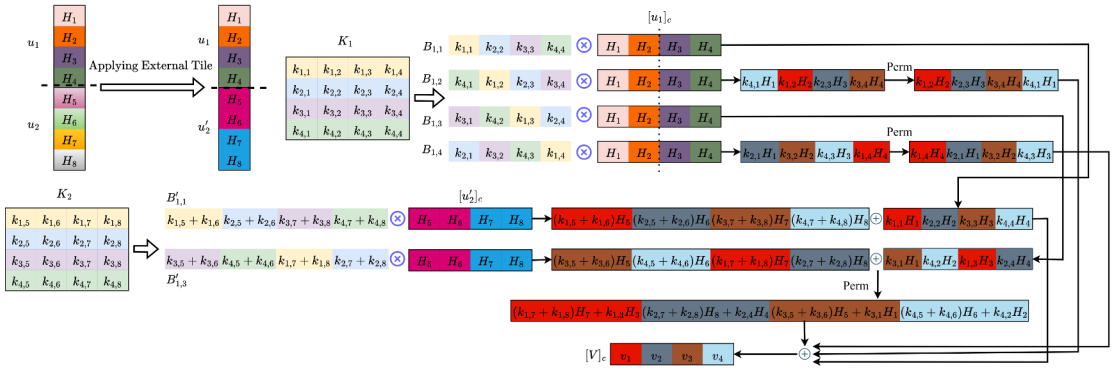


Figure 5: MIMO convolution applied with the External Tile structure

challenge for effectively applying the tile structures.

Moreover, the sensitivity to tile structures differs across various types of convolution layers. For instance, within ResNet50's residual block, 1×1 convolution layers predominantly perform dimensional reduction and expansion before and after the 3×3 convolution layers, whereas the 3×3 convolution layers focus on capturing spatial information [42], [38]. Implementing tile structures on the 1×1 convolution layers reduces the information capacity for the subsequent 3×3 convolution layers.

Therefore, it is important to find the optimal positions to apply the tile structures. To this end, we develop a tile position locator based on the OTO technique [43] to identify optimal candidate channels for tiles. The key idea is to compute a score for each input channel for a layer. Input channels with scores below a threshold are identified as candidates for applying the tile structures.

Algorithm 1 describes the tile position locator to identify the optimal positions to apply a tile structure for a convolution layer. It first analyzes the SI cost for the Perm operation to determine the tile structure type to be applied for the convolution layer. If the inference cost of the external perm is higher than the internal perm, denoted as $\frac{t_{\text{in}} C_i}{C_o} \leq \frac{t_{\text{ex}}(f^2-1)}{C_n-1}$, the External Tile structure will be applied; otherwise, the Internal Tile structure will be used. Here, t_{in} and t_{ex} represent the average time for an internal and external perm operation in a convolution operation, respectively. C_n represents the number of input channels that are packed into one ciphertext.

Next, the tile position locator identifies the optimal positions to apply the tile structure. If the selected structure is the External Tile structure, the tile position locator partitions the C_i input channels sequentially into m distinct groups, where $m = \frac{C_i}{C_n}$. Each group is treated as though it is encoded into one separate ciphertext. Subsequently,

the External Tile is applied to every group. If the selected structure is the Internal Tile structure, the Internal Tile is applied to every input channel. The tile position locator then estimates the stochastic gradient, denoted as $\nabla\tilde{f}(\mathbf{W})$, and computes the stochastic sub-gradient, $\mathbf{g}$, defined as $\mathbf{g} = \nabla\tilde{f}(\mathbf{W}) + \lambda\zeta(\mathbf{W})$, and updates the model weights. The updated weights $\mathbf{W}' = \{w'_{i,j}\}$ $(1 \le i \le C_i, 1 \le j \le C_o \times f^2)$ for the layer with the applied tile structure are calculated as $\mathbf{W}' = \mathbf{W} - lr \times \mathbf{g}$ for the subsequent iteration. Here, λ is the weighting coefficient for the sub-gradient $\zeta(\mathbf{W})$, calculated using a mixed l_1/l_2 norm on $\mathbf{W}$, as described in [43]. $\mathbf{W} = \{w_{i,j}\}$ $(1 \le i \le C_i, 1 \le j \le C_o \times f^2)$ represents the pre-trained weight of a layer in the original model, where C_o and C_i are the number of output and input channels, respectively, and f represents the filter size. The term lr denotes the pre-defined learning rate.

Let $\mathbf{W}_n = \{w_{n,p}\}$ $(p \in C_o \times f^2)$ denote the n-th row of $\mathbf{W}$. $\|\mathbf{W}_n\|^2$ provides an estimation of the importance for the n-th input channel. The channel applicability score $\{s_n\}$ for the input channel n $(1 \le n \le C_i)$ is calculated using the following equation:

$$s_n = \lg\left(\left|\frac{1}{\|\mathbf{W}_n\|^2}\sum_{j=1}^{C_o\times f^2} w'_{n,j}w_{n,j}\right|\right), \tag{2}$$

where lg is the logarithm function that is used to scale down the value of the applicability score. Here, the inner product between $\mathbf{W}$ and $\mathbf{W}'$ estimates the similarities between the original model's weights and the weights after applying the tile structure to the input channel n. This measurement assesses how closely the updated weights maintain the directional and magnitude characteristics of the original model's weights.

With these computed scores, the tile position locator can effectively determine the optimal positions for applying the tile structures. For example, if the tile position locator selects the External Tile structure, it sums the applicability scores s_n of all input channels in group m, to get the group's applicability score, denoted as $\gamma_m = \sum_{n\in m} s_n$. The tile position locator then applies the External Tile to the m-th group if $\gamma_m \le \epsilon$. If the tile position locator chooses the Internal Tile structure, it applies the Internal Tile to the n-th input channel if $s_n \le \epsilon$.

Considering the functional differences between the 3×3 convolution layer and the 1×1 convolution layer, the tile position locator adjusts the threshold for the 1×1 convolution layer as $\epsilon' = \frac{\alpha * C_o}{C_i} * \epsilon$, which is then set as the new ϵ. Here, ϵ represents the pre-defined threshold for determining whether a tile structure should be applied to a group of input channels. α represents a control weight that adjusts according to the layer's position within the model, and $\frac{C_o}{C_i}$ acts as the dimension scaling factor for the 1×1 convolution layer. The control weight α is calculated as $\alpha = e^{\beta*(\Delta h)}$, with β representing the decay coefficient and Δh indicating the distance from the current layer to the final output. This adaptive mechanism allows α to gradually reduce the impact of the applied tile structures as the distance from the final output layer increases. Such adaptive adjustments refine the input structuring process by considering the type, parameters, and positioning of layers within the model.

Algorithm 1: Tile Position Locator for a Convolution Layer

Input: Pre-trained weight $\mathbf{W}$ for a convolution layer
Output: Applicability Score S

Select the External Tile structure if $\frac{t_{\text{in}}C_i}{C_o} \le \frac{t_{\text{ex}}(f^2-1)}{C_n-1}$, otherwise select the Internal Tile
if *the selected structure is External Tile* **then**
  Partition the C_i input channels sequentially into $m = \frac{C_i}{C_n}$ distinct groups
  Apply External Tile on every group
else
  Apply Internal Tile on every input channel
end
Compute a stochastic sub-gradient: $g = \nabla\tilde{f}(\mathbf{W}) + \lambda\zeta(\mathbf{W})$
Update the layer weight: $\mathbf{W}' = \mathbf{W} - lr \times g$
for $1 \le n \le C_i$ **do**
  Estimate the applicability score for input channel n: $s_n = \lg(\left|\frac{1}{\|\mathbf{W}_n\|^2}\sum_{j=1}^{C_o\times f^2} w'_{n,j}w_{n,j}\right|)$
end
if *this is a* 1×1 *convolution layer* **then**
  Compute the adjusted threshold: $\epsilon' = \frac{\alpha * C_o}{C_i} * \epsilon$
  Set $\epsilon = \epsilon'$
if *the selected structure is External Tile* **then**
  Estimate the channel applicability score for the input channel group m: $\gamma_m = \sum_{n\in m} s_n$
  Apply the External Tile structure to input channel group m if $\gamma_m \le \epsilon$
else
  Apply the Internal Tile structure to input channel n if $s_n \le \epsilon$
end

At last, we can apply the tile position locator on each convolution layer of a model to find optimal positions for tiles and apply the tile structures at those layers. Next the model is fine-tuned to recover its accuracy. We use the Knowledge Distillation (KD) method [37] to guide this fine-tuning process.

3.3. Convolution Protocol with Tile structure

We now present the process for evaluating convolution after applying the Tile structure. After the server and client have predefined the Tile structure positions offline, the server shares a list, I, with the client, indicating the input channel positions where the Tile structure will be applied in each convolution layer. The client pre-processes its input $u = \{u_i\}$ $(0 \le i \le C_i)$ by applying the Tile structure to every input channel u_i if $i \in I$. Then, the

client encrypts u as $[u]_c$, which is sent to the server. The server modifies its kernel k based on the predefined Tile structure type and position list I and conducts convolution computation with $[u]_c$ to obtain the encrypted output $[v]_c$, where $[v]_c = [u]_c * k \oplus r$. Here, r is a random vector generated by the server. Finally, the server sends $[v]_c$ to the client and outputs the random vectors r. The client then decrypts $[v]_c$ and output v. Compared to CryptFlow2, TILE modifies only the encoding method for the convolution input and its corresponding kernel (as shown in Figures 3 and 5), without altering CryptFlow2's overall convolution computation logic. The security analysis of TILE is presented in the following section.

3.4. Security Analysis

The primary goal of TILE is to eliminate HE operations in convolution computations within the SI framework by introducing two novel input structures, while preserving the original computation logic. By determining the suitability of each convolution layer in models such as VGG and ResNet, TILE reduces the need for HE operations in convolutions and minimizes associated OT-based non-linear operations, like ReLU. TILE, like most SI platforms, does not aim to completely obscure the network architecture but focuses on ensuring the protection of critical model parameters, such as weights, filter sizes, and stride within convolution layers, as well as the client input data. Built on the underlying SI framework such as CryptFlow2 [13], which has already been proven secure under the semi-honest threat model, TILE leverages the semantic security of the PHE algorithm [24] to speed up linear computations in the underlying SI framework, providing a comprehensive security approach.

Beyond the underlying SI framework, TILE does have additional information exchange between the server and client that could pose a risk of information leakage to both the server and the client. First of all, from the client's point of view, the server knows the Tile structure positions. Hence, one might think the server could infer that the elements in a Tile structure of the input data from the client have similar values, which could form a covert channel, raising a risk for potential information leakage of the client input data to the server. Nevertheless, it is important to note that in the inference phase, the TILE operation is inherently deterministic, regardless of the actual pixel values of a given input data. More specifically, for a given layer, the grouping of channels or elements into a Tile structure is predetermined before the inference phase. During inference, the server applies the Tile structure to the input data or feature maps based on these predefined Tile positions, regardless of the actual input values. As a result, the server actually cannot infer the values are similar, simply because those values are in a Tile. In fact, for a specific input data, the actual values in a Tile can be quite different.

Second, from the server's point of view, the need to share the predefined Tile structure positions with the client could introduce a risk of information leakage of model parameters to the client. Nevertheless, in practice, the possibility that the client infers the model parameters, such as the convolution filter values, based on the Tile structure and its predefined positions is very low. Let us take the convolution filter values as an example. To infer the filter values, the client would need to know the output values of each convolution layer, which is, however, rather challenging. First, as described in Section 3.3, the server only sends the output share of the convolution result v to the client, which is masked by a random vector r by the server. Second, the server can randomize the order of output channels in the previous convolution, further preventing the client from associating to specific outputs with corresponding inputs.

When these techniques are combined, the probability that the client successfully infers the output of a convolution layer is given by

$$Pr_{\text{success}} = \frac{1}{\binom{C_i}{d}} \times \left(\frac{1}{d!}\right) \times \left(\frac{1}{2^\ell}\right)^{C_i \times u_w \times u_h}.$$

The first term is the probability that the client can successfully guess which d out of C_i channels have been applied with the Tile structure by the server. The second term indicates the probability that the client successfully guesses the order of the d channels. The last term is the probability that the client successfully guess the ℓ-bit mask for every pixel value in each channel, where u_w and u_h are the height and width of a channel.

As ℓ, C_i and d are generally large, this probability is a negligible value. For instance, in a scenario where $C_i = 64$, $d = 8$, $\ell = 64$, and $u_w = u_h = 4$, the success probability $Pr_{success} \approx 2^{-65,588}$, which is significantly smaller than a typical statistical security requirement of $Pr_{success} \leq 2^{-40}$. Consequently, the client would find it practically impossible to infer the model parameters of the server.

3.5. Complexity Analysis

We focus on the number of Internal Perm, Mult, and Add operations as they dominate the computation of SI frameworks. The underlying SI framework is assumed as the state-of-the-art CrypTFlow2. With CrypTFlow2, for a 3×3 convolution layer using the Grouped Out-Rot MIMO, applying the Internal Tile structure reduces the number of Internal Perm operations from $\frac{8C_i}{C_n}$ to $\frac{(8-4s)C_i}{C_n}$ and the number of Mult operations from $\frac{9C_iC_o}{C_n}$ to $\frac{(9-4s)C_iC_o}{C_n}$. On the other hand, applying the External Tile structure maintains the number of Internal Perm operations but reduces the number of Mult operations from $\frac{9C_iC_o}{C_n}$ to $\frac{9(2-s)C_iC_o}{2C_n}$. For a 1×1 convolution layer using the Grouped Out-Rot MIMO, applying the External Tile structure reduces the number of Mult operations from $\frac{C_iC_o}{C_n}$ to $\frac{(2-s)C_iC_o}{2C_n}$.

Table 1 illustrates the complexity of convolution computations for CrypTFlow2 before and after applying TILE, where s is the applying ratio for tile structures, n_i is the size of input data, C_i is the number of input channels, C_o is the number of output channels, C_n is the number of channels that can be packed into one ciphertext, and the tile size for Internal Tile and group size for External Tile are

TABLE 1: Complexity Comparison of Convolution

Method	Grouped Out-Rot MIMO[13]					
	# Internal Perm		# Mult		# Add	
Tile Type	External Tile					
	Before TILE	After TILE	Before TILE	After TILE	Before TILE	After TILE
1X1 Convolution	0	0	$\frac{C_i C_o}{C_n}$	$\frac{(2-s)C_i C_o}{2C_n}$	$\frac{C_o(C_i-1)}{C_n}$	$\frac{(2-s)C_i C_o}{2C_n} - \frac{C_o}{C_n}$
3X3 Convolution	$\frac{8C_i}{C_n}$	$\frac{8C_i}{C_n}$	$\frac{9C_i C_o}{C_n}$	$\frac{9(2-s)C_i C_o}{2C_n}$	$\frac{C_o(9C_i-1)}{C_n}$	$\frac{9(2-s)C_i C_o}{2C_n} - \frac{C_o}{C_n}$
Tile Type	Internal Tile					
	Before TILE	After TILE	Before TILE	After TILE	Before TILE	After TILE
3X3 Convolution	$\frac{8C_i}{C_n}$	$\frac{(8-4s)C_i}{C_n}$	$\frac{9C_i C_o}{C_n}$	$\frac{(9-4s)C_i C_o}{C_n}$	$\frac{C_o(9C_i-1)}{C_n}$	$\frac{(9-4s)C_i C_o}{C_n} - \frac{C_o}{C_n}$

TABLE 2: Complexity Comparison of Non-Linear Computation

#.Non-Linear Operation	CrypTFlow2	TILE-Internal	TILE-External
ReLU/Truncation	n_i	$(1-\frac{3s}{4})n_i$	$(1-\frac{s}{2})n_i$

both assumed to be 2. Table 2 illustrates the computational complexity associated with ReLU and truncation operations. For both operations, applying the Internal Tile structure reduces the number of operations from n_i to $(1-\frac{3s}{4})n_i$, while applying the External Tile structure reduces it to $(1-\frac{s}{2})n_i$.

TABLE 3: Performance Results on Cifar10 and Tiny-ImageNet

Dataset	Cifar10	Tiny-ImageNet
Model	VGG16	ResNet50
Operation Reduction		
#. Internal Perm	1224/1672 26.79%↓	1500/2768 45.81%↓
#. Mult	282.422K/635.118K 55.53%↓	353.504K/779.39K 54.64%↓
#. ReLU	568.32K/1.08M 49.20%↓	7.50M/11.08M 32.29%↓
Model Accuracy		
Baseline Acc	94.50%	66.07%
Our Acc	94.66%	66.10%

4. Evaluation

In this section, we evaluate TILE's performance when applying it to CrypTFlow2 [13], which is a state-of-the-art SI framework. Note that TILE is also applicable to other SI frameworks based on PHE, such as Gazelle[15], Delphi [17], and Gala[18]. We use the Secure and Correct Inference (SCI) library within EzPC [13], [44] [2] to implement the HE-based privacy-preserving inference for convolution computation. We set the input bit-length to 37-bits and the scale to 12. In our experiments, we utilize two widely used CNN models, VGG16 and ResNet50, and evaluate them on two mainstream datasets: Cifar10 [2] and Tiny-ImageNet [45]. We use pretrained models sourced from [32]. The decay coefficient in the TILE position locator is set to 0.1. Fine-tuning of the models is performed using an SGD optimizer with a weight decay of 5e-4, momentum of 0.9, and an initial learning rate of 0.01, which is reduced by a factor of 10 at epochs 30 and 45. The batch size is set to 64, and the number of fine-tuning epochs is 60. We employ the KD scheme in [37] to guide the fine-tuning process. For models trained on Cifar10, the KD temperature is set to 1, and for models trained on Tiny-ImageNet, the KD temperature is set to 4. We also conduct experiments to apply TILE to models that have already been compressed, by either model pruning or model distillation, and demonstrate that TILE can further significantly improve their computation time.

All experiments are conducted on a Lambda Vector workstation, which is equipped with an NVIDIA RTX A6000 48GB GPU, an AMD Ryzen Thread ripper PRO 3995WX 64-Cores CPU, and 512GB RAM. The performance metric considered in our work is the end-to-end running time, which includes the time for data transmission through networks with the bandwidth of **125 Mbps (Mobile)**, **400 Mbps (WAN)**, and **3 Gbps (LAN)**, and the round-trip latency of approximately 30 ms, 40 ms, and 0.3 ms, respectively. These settings are based on the same network configurations used in [13], [19] and real-world network performance data reported by Ookla[3]. To simulate different network conditions, we utilize the EMP Toolkit [46] along with the Linux Traffic Control and employ 16 threads to evaluate each model's SI performance.

TILE Performance Compared with Baseline: In the ensuing discussion, the original CrypTFlow2 performance is assumed as the baseline. Tables 3 and 4 illustrate the performance when applying TILE to CrypTFlow2. The data in the table is in the form x/y, where x indicates the performance after applying TILE and y indicates the performance of the baseline. For instance, for VGG16 with Cifar10, the number of Internal Perm is 1224/1672, which means after applying TILE, there are 1224 Internal Perm operations, while before applying TILE, there are 1672 Internal Perm operations. The percentage in the table indicates the time reduction percentage. We did not include the numbers for the Add operation in our analysis, as its contribution to the overall computational cost is negligible.

The performance gain of TILE stems from its capability in identifying and eliminating Perm, Mult, and corresponding non-linear operations. Experimental results have shown that TILE performs very well on smaller models such as VGG16, largely due to the inherent redundancy in their input feature maps, which allows significant input structure

2. https://github.com/mpc-msri/EzPC/tree/master/SCI

3. https://www.speedtest.net/global-index/united-states

TABLE 4: End-to-End Computation Time on Cifar10 and Tiny-ImageNet in Different Network Setting

Dataset	Cifar10		
Model	VGG16		
Network	Mobile	WAN	LAN
Total Time (s)	193.29/374.46 48.38% ↓	153.19/300.27 48.98% ↓	123.78/255.58 51.57% ↓

Dataset	Tiny-ImageNet		
Model	ResNet50		
Network	Mobile	WAN	LAN
Total Time (s)	1100.26/1610.81 31.70% ↓	527.58/800.15 34.06% ↓	267.84/455.48 41.20% ↓

optimization without compromising model accuracy. For instance, applying TILE to VGG16 running on the Cifar10 dataset resulted in significant operation reduction: a 26.79% decrease in Internal Perms operations, a 55.53% decrease in Mult operations, and a 49.20% decrease in ReLU operations. Such operation reduction translates into substantial time savings—48.38% on the Mobile, 48.98% on WAN, and 51.57% on LAN settings, respectively. TILE is also effective in more complex models like ResNet50. For the Tiny-ImageNet dataset, TILE achieves a 45.81% reduction in Internal Perms operations, a 54.64% reduction in Mult operations, and a 32.29% reduction in ReLU operations. This efficiency gain results in time saving of 31.70% on Mobile, 34.06% on WAN, and 41.20% on LAN settings, respectively.

As shown in Tables 5 and 6, TILE remains effective for pruned models that already have high computation cost reduction by model pruning using the MOSAIC strategy [32]. For example, a pruned ResNet50, which has already achieved a 57.62% time reduction, still gets an additional reduction of 40.34% Internal Perms, 51.37% Mult operations, and 28.02% ReLU operations. This leads to a time reduction of 42.46% on Mobile, 36.05% on WAN, and 25.90% on LAN settings, respectively. The performance gain of TILE is smaller when applied to a pruned VGG16 architecture on Cifar10. This is due to the small scale of the input feature maps in the intermediate convolution layers; thus the model needs more input feature map details in those layers, thereby reducing the redundancy available for TILE to exploit effectively. Nevertheless, TILE still manages to reduce 13.36% Internal Perm operations, 14.60% Mult operations, and 33.78% ReLU operations, achieving a time saving of 24.61% on Mobile, 20.10% on WAN, and 19.96% on LAN settings, respectively.

Similarly, TILE can be applied to models compressed through model distillation to further enhance SI performance. Due to the space limitation, the performance evaluation results are presented in Appendix A.

Next, we examine the trade-off between the time saving and accuracy loss. We apply the Tile structure to all input channel in every convolution layer, to maximize the time saving. Table 7 illustrates the time savings. For instance, when applied to VGG16 on the Cifar10 dataset, TILE reduces the computation time by 55.97% on Mobile, 53.06% on WAN, and 54.42% in LAN, which is a further time

TABLE 5: Performance Results for the Pruned Models

Dataset	Cifar10	Tiny-ImageNet
Model	VGG16	ResNet50
Cost Reduction (by Prunning)	71.34%	57.62%
Operation Reduction		
#. Internal Perm	804/928 13.36%↓	692/1160 40.34%↓
#. Mult	146.65K/171.73K 14.60%↓	124.94K/256.92K 51.37%↓
#. ReLU/Truncation	395.59K/597.38K 33.78%↓	5.41M/7.51M 28.02%↓
Model Accuracy		
Baseline Acc	94.41%	65.55%
Our Acc	94.47%	65.75%

TABLE 6: End-to-End Time Performance for the Pruned Models

Dataset	Cifar10		
Model	VGG16		
Network	Mobile	WAN	LAN
Overall Cost(s)	111.17/147.47 24.61% ↓	84.13/105.30 20.10% ↓	58.62/73.25 19.96% ↓

Dataset	Tiny-ImageNet		
Model	ResNet50		
Network	Mobile	WAN	LAN
Overall Cost(s)	577.918/1004.33 42.46% ↓	282.04/441.00 36.05% ↓	144.37/194.83 25.90% ↓

saving compared with the ones in Table 4, but at the expense of a 1.28% accuracy drop. For more complex models like ResNet50, TILE achieves time reductions of 47.40% on Mobile, 47.33% on WAN and 51.92% on LAN, which are a significant further time saving compared with the ones in Table 4, with just a 0.40% accuracy drop. For pruned models, such as pruned ResNet50 on Tiny-Imagenet, TILE achieves a 54.49% time reduction on Mobile, 47.39% on WAN, and 36.08% on LAN, which are also a significant time saving compared with the results in Table 6, with a slightly higher accuracy drop, 1.23%. Similarly, for pruned VGG16 on Cifar10, TILE reduces time by 47.52% on Mobile, 42.09% on WAN, 40.31% on LAN, which are also a significant time saving compared with the results in Table 6, with a little higher accuracy drop, 3.69%. These results demonstrate that TILE can further reduce computational costs substantially, if a slight model accuracy drop is acceptable.

TILE Layer-Wise Performance Breakdown: Figures 6 and 7 illustrate the layer-wise breakdown of performance for the TILE-optimized model on the Mobile network setting. In these figures, we illustrate the actual time cost of SI (represented as bars) for both the original model and the TILE-optimized model in the CrypTFlow2 framework, along with the corresponding time-saving ratio (shown as lines) for each convolution layer within the models. In both tested models, the first convolution layer usually interacts with the input image directly and demonstrates a pronounced sensitivity to TILE due to its minimal redundancy. Nevertheless, as the computational cost associated with this convolution layer constitutes a minor portion of the overall computational

TABLE 7: Performance Results for the Efficiency and Accuracy Trade-off

Dataset	Cifar10		Tiny-ImageNet	
Model	VGG16	Prunned VGG16	ResNet50	Prunned ResNet50
Accuracy	93.22%/94.50% 1.28% ↓	90.72%/94.41% 3.69% ↓	65.67%/66.07% 0.40% ↓	64.32%/65.55% 1.23% ↓
Network	Overall Cost (s)			
Mobile	164.87/374.46 55.97% ↓	77.39/147.47 47.52% ↓	847.27/1610.81 47.40% ↓	457.06/1004.33 54.49% ↓
WAN	140.96/300.27 53.06% ↓	60.98/105.30 42.09% ↓	421.42/800.15 47.33% ↓	232.01/441.00 47.39% ↓
LAN	116.51/255.58 54.42% ↓	43.72/73.25 40.31% ↓	219.00/455.48 51.92% ↓	124.53/194.83 36.08% ↓

time, we choose not to apply the tile structure to this layer.

In the case of the VGG16 architecture on the Cifar10 dataset, 10 out of 13 convolution layers achieve a time reduction exceeding 34.03%, as illustrated in Figure 6(a). Conversely, with the ResNet50 model on the Tiny-ImageNet dataset, approximately 22 out of 49 convolution layers attain a time reduction of more than 22.12%. Of these, 12 layers achieve time reductions exceeding 41.48%, as demonstrated in Figure 7(a).

When applying TILE to pruned models, TILE can further enhance the pruned model's efficiency on the Tiny-ImageNet dataset, particularly within the later layers, as illustrated in Figure 7(b). For more complex models like pruned ResNet50, about 43% of layers achieve a time reduction exceeding 21%, and 24% of these layers surpass a reduction of 42%. On the other hand, when applying TILE to the pruned VGG16 architecture on the Cifar10 dataset, TILE shows less improvement, resulting in only a small reduction in the overall inference time across all layers, as depicted in Figure 6(b). In this case, only about 23% of the convolution layers can achieve a reduction in time of up to 17%.

5. Conclusions and Future Directions

In this paper, we introduce an innovative input structure optimization system known as TILE to effectively streamline the computation process for SI frameworks. The outcome of TILE is an optimized model that retains its accuracy while achieving a reduction in HE operations. The application of TILE was demonstrated on widely used architectures, VGG16 and ResNet50, using well-known datasets such as Cifar10 and Tiny-ImageNet. Our experiments show that TILE efficiently reduces HE Perm, Mult, Add, and corresponding non-linear operations required for the original model's SI without compromising accuracy. Furthermore, the synergistic application of TILE with HE-friendly model pruning and model distillation led to additional reductions in HE Perm, Mult, and Add operations within pruned AI models. TILE introduces a promising avenue for significantly reducing the time required for SI. Its impact extends toward enhancing the feasibility of SI in practical settings. As such, TILE not only contributes to the current state of research but also offers insights that could guide future investigations in this domain.

TILE demonstrates strong generalizability across both small and large CNN architectures, including popular models such as VGG16 and ResNet50. Given the structural consistency of CNNs, TILE effectively optimizes input structures across different models. However, newer architectures like EfficientNet [47], which employs compound scaling, and Vision Transformers [48], which abandon convolution entirely, present unique structural characteristics. This presents a valuable direction for future work, as adapting TILE to these emerging architectures could reveal unique optimization challenges and opportunities.

Acknowledgment

The authors would like to express their gratitude to the anonymous shepherd and reviewers for their constructive comments and also appreciate the lab members for their assistance and collaboration on this work. The work of Q. Zhang was supported in part by the National Natural Science Foundation of China under Grant 62302067. The work of H. Wu was supported in part by the NSF under Grant OAC-2320999, CNS-2120279, IIS-2236578, DGE-2336109, CNS-2413009, and SaTC-2439013.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[3] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proc. IEEE CVPR*, 2015, pp. 815–823.

[4] Q. Fan and J. Yang, "A denoising autoencoder approach for credit risk analysis," in *Proc. ICCAI*, 2018, pp. 62–65.

[5] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber, "Using deep learning to enhance cancer diagnosis and classification," in *Proc. ICML*, vol. 28. ACM New York, NY, USA, 2013, pp. 3937–3949.

[6] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad, "Rafiki: Machine learning as an analytics service system," *Proc. VLDB Endow.*, vol. 12, no. 2, p. 128–140, oct 2018. [Online]. Available: https://doi.org/10.14778/3282495.3282499

[7] H. S. Maghded, K. Z. Ghafoor, A. S. Sadiq, K. Curran, D. B. Rawat, and K. Rabie, "A Novel AI-enabled Framework to Diagnose Coronavirus COVID-19 using Smartphone Embedded Sensors: Design Study," in *Proc. IEEE IRI*, 2020, pp. 180–187.

[8] D. Demmler, T. Schneider, and M. Zohner, "ABY-a framework for efficient mixed-protocol secure two-party computation." in *Proc. NDSS*, 2015.

[9] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation." in *Proc. USENIX Security*, 2021, pp. 2165–2182.

[10] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC*, 2018, pp. 35–52.

[11] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. ICML*. PMLR, 2016, pp. 201–210.

[12] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *Proc. IEEE S&P*, 2020, pp. 336–353.

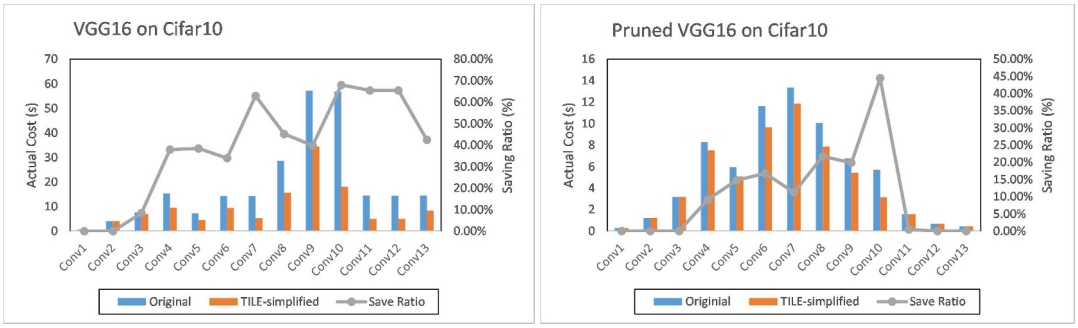
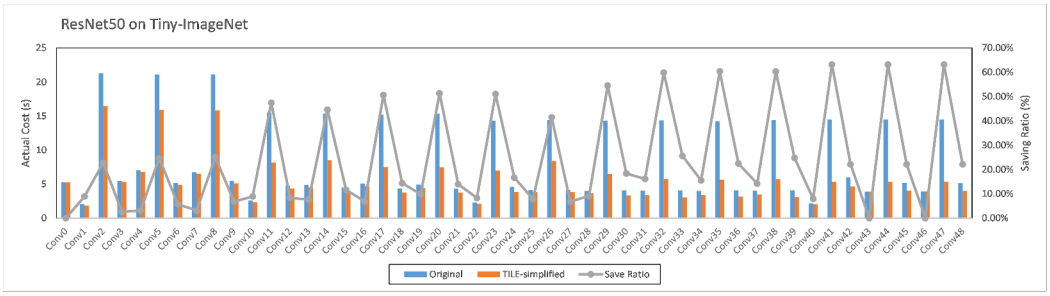
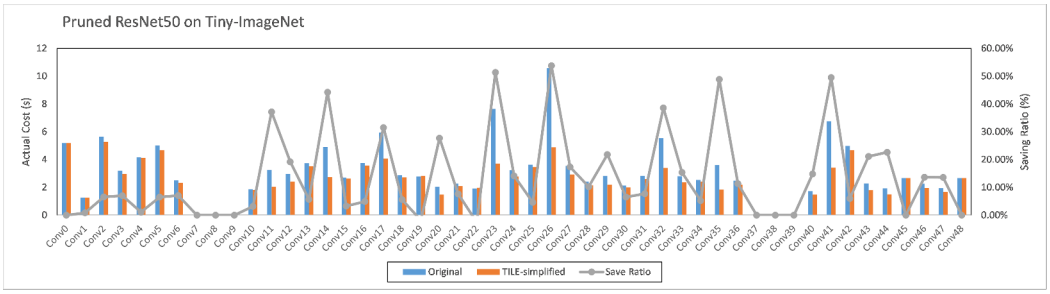


Figure 6: Layer-Wise Performance breakdown for Cifar10 dataset on (a) VGG16, (b) Pruned VGG16 model.



(a)



(b)

Figure 7: Layer-Wise Performance breakdown for Tiny-ImageNet dataset on (a) ResNet50, (b) Pruned ResNet50 model.

[13] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proc. ACM CCS*, 2020, pp. 325–342.

[14] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *Proc. ARES*, 2020, pp. 1–10.

[15] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proc. USENIX Security*, 2018, pp. 1651–1669.

[16] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proc. IEEE S&P*, 2017, pp. 19–38.

[17] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: a cryptographic inference system for neural networks," in *Proc. USENIX Security*, 2020, pp. 27–30.

[18] Q. Zhang, C. Xin, and H. Wu, "Gala: Greedy computation for linear algebra in privacy-preserved neural networks," in *Proc. NDSS*, 2021.

[19] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure {Two-Party} deep neural network inference," in *Proc. USENIX Security*, 2022, pp. 809–826.

[20] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. Asia CSS*, 2018, pp. 707–721.

[21] A. Patra and A. Suresh, "BLAZE: blazing fast privacy-preserving machine learning," in *Proc. NDSS*, 2020.

[22] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "{XONN}:{XNOR-based} oblivious deep neural network inference," in *Proc. USENIX Security*, 2019, pp. 1501–1518.

[23] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Archive*, 2012.

[24] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in lwe-based homomorphic encryption," in *Proc. PKC*. Springer, 2013, pp. 1–13.

[25] A. C. Yao, "Protocols for secure computations," in *Proc. FOCS*. IEEE, 1982, pp. 160–164.

[26] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proc. ACM CCS*, 2012, pp. 784–796.

[27] G. Brassard, C. Crépeau, and J.-M. Robert, "All-or-nothing disclosure of secrets," in *Proc. Eurocrypt*. Springer, 1986, pp. 234–238.

[28] V. Kolesnikov and R. Kumaresan, "Improved ot extension for transferring short secrets," in *Proc. CRYPTO 2013*. Springer, 2013, pp. 54–70.

[29] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *Proc. ACM CCS*, 2019, pp. 291–308.

[30] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[31] Y. Cai, Q. Zhang, R. Ning, C. Xin, and H. Wu, "Hunter: He-friendly structured pruning for efficient privacy-preserving deep learning," in *Proc. ACM Asia CCS*, 2022, pp. 931–945.

[32] ——, "Mosaic: A prune-and-assemble approach for efficient model pruning in privacy-preserving deep learning," in *Proc. ACM Asia CCS*, 2024, p. 1034–1048.

[33] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. ECCV*. Springer, 2014, pp. 818–833.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[36] S. Kundu, S. Lu, Y. Zhang, J. T. Liu, and P. A. Beerel, "Learning to linearize deep neural networks for secure and efficient private inference," in *Proc. ICLR*, 2023.

[37] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, 2016, pp. 770–778.

[39] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. ACM SIGSAC*, 2017, pp. 619–631.

[40] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[41] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Proc. NeurIPS*, vol. 28, 2015.

[42] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE CVPR*, 2015, pp. 1–9.

[43] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, and X. Tu, "Only train once: A one-shot neural network training and pruning framework," *Proc. NeurIPS*, vol. 34, pp. 19 637–19 651, 2021.

[44] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: Programmable and efficient secure two-party computation for machine learning," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 496–511.

[45] Computer Vision Lab at Stanford University, "Tiny imagenet dataset," 2015. [Online]. Available: http://cs231n.stanford.edu/

[46] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[47] T. Mingxing and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, vol. 97. PMLR, 2019, pp. 6105–6114.

[48] D. Alexey, B. Lucas, K. Alexander, W. Dirk, Z. Xiaohua, U. Thomas, D. Mostafa, M. Matthias, H. Georg, G. Sylvain, U. Jakob, and H. Neil, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

Appendix A.
Evaluation performance for TILE with Model Distillation

Model distillation (MD) is a crucial model compression technique that accelerates the inference process by reducing model size. TILE can be applied to models compressed through MD, allowing for seamless integration and further performance improvements. In our experiments, we utilize two widely used CNN models—VGG16 and ResNet50 as teacher models, and VGG11 and ResNet18 as student models. The distillation process follows the method outlined in [37], and the evaluations are conducted on two widely adopted datasets: Cifar10 [2] and Tiny-ImageNet [45].

In the ensuing discussion, the original CrypTFlow2 performance for MD compressed model is assumed as the baseline. As shown in Tables 8 and 9, TILE remains effective for MD compressed models that already have high computation cost reduction by reducing the model size. For example, an MD-compressed ResNet18, which has already achieved a 76.67% time reduction compared to ResNet50, still achieves an additional reduction of 40.49% Internal Perms, 42.76% Mult operations, and 48.53% ReLU operations, after applying TILE. This leads to a time reduction of 45.95% on Mobile, 45.15% on WAN, and 47.10% on LAN settings, with only a 0.67% accuracy drop. A similar outcome is observed when applying TILE to an MD-compressed VGG11 architecture on Cifar10, where TILE reduces Internal Perm operations by 19.75%, Mult operations by 68.83%, and ReLU operations by 52.94%, leading to time savings of 47.52% on Mobile, 51.52% on WAN, and 51.57% on LAN settings, with literally no accuracy loss.

TABLE 8: Performance Results for the MD Models

Dataset	Cifar10	Tiny-ImageNet
MD Model	VGG11	ResNet18
Cost Reduction (by MD)	36.25%	76.67%
Operation Reduction		
#. Internal Perm	520/648 19.75% ↓	1552/2608 40.49% ↓
#. Mult	80.436K/258.036K 68.83% ↓	262.186K/458.051K 42.76% ↓
#. ReLU	262.144K/557.056K 52.94% ↓	1.147M/2.23M 48.53% ↓
Model Accuracy		
Baseline Acc	94.08%	66.17%
Our Acc	93.97%	65.50%

TABLE 9: End-to-End Time Performance for the MD Models

Dataset	Cifar10		
Model	VGG11		
Network	Mobile	WAN	LAN
Total Time (s)	24.61/46.9 47.52% ↓	23.79/46.77 51.25% ↓	22.76/46.69 51.57% ↓

Dataset	Tiny-ImageNet		
Model	ResNet18		
Network	Mobile	WAN	LAN
Total Time (s)	214.260/396.396 45.95% ↓	111.075/202.515 45.15% ↓	56.734/107.257 47.10% ↓