Efficient ML Models for Practical Secure Inference

Vinod Ganesan[1,2], **Anwesh Bhattacharya**[1], **Pratyush Kumar**[1,2], **Divya Gupta**[1], **Rahul Sharma**[1], **Nishanth Chandran**[1]

[1] Microsoft Research, India
[2] IIT Madras
t-viganesan, t-anweshb, pratykumar, divya.gupta, rahsha, nichandr@microsoft.com

Abstract

ML-as-a-service continues to grow, and so does the need for very strong privacy guarantees. Secure inference has emerged as a potential solution, wherein cryptographic primitives allow inference without revealing users' inputs to a model provider or model's weights to a user. For instance, the model provider could be a diagnostics company that has trained a state-of-the-art DenseNet-121 model for interpreting a chest X-ray and the user could be a patient at a hospital. While secure inference is in principle feasible for this setting, there are no existing techniques that make it practical at scale. The CrypTFlow2 framework provides a potential solution with its ability to automatically and correctly translate clear-text inference to secure inference for arbitrary models. However, the resultant secure inference from CrypTFlow2 is impractically expensive: Almost 3TB of communication is required to interpret a single X-ray on DenseNet-121.

In this paper, we address this outstanding challenge of inefficiency of secure inference with three contributions. First, we show that the primary bottlenecks in secure inference are large linear layers which can be optimized with the choice of network backbone and the use of operators developed for efficient clear-text inference. This finding and emphasis deviates from many recent works which focus on optimizing non-linear activation layers when performing secure inference of smaller networks. Second, based on analysis of a bottlenecked convolution layer, we design a X-operator which is a more efficient drop-in replacement. The X-operator combines various ideas which we found to be efficient for secure inference: factorizing linear operations along with using shuffle operations and larger proportion of additions, both of which are "free" (i.e., require no communication) in secure inference. Third, we show that the fast Winograd convolution algorithm further improves efficiency of secure inference. We add Winograd algorithm support to the Athos compiler frontend of CrypTFlow2, thereby automatically improving efficiency for most deep networks. In combination, these three optimizations prove to be highly effective for the problem of X-ray interpretation trained on the CheXpert dataset: Relative to state-of-the-art models, we identify a model with about $8\times$ lower communication cost with a negligible drop in AUC of 0.006, and another model with over $30\times$ lower cost with a drop in AUC of 0.02. We believe that inference with very strong privacy guarantees can be made practical with such efficient ML models.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Introduction

Often large Machine Learning (ML) models are deployed on cloud computers which run inference on user data, a template loosely called ML-as-a-service (MLaaS). A growing concern with this template is the need for guaranteeing user privacy. This is particularly important in domains such as healthcare where user privacy is paramount and often legally binding (HIPAA 2012; Soin et al. 2021). In addition, ML models for healthcare, such as models for prognosis, have high commercial value given the challenges with obtaining training data and thus need to be protected. One of the approaches to interface private data with protected models is *secure inference* with cryptographically-secure 2-party computation (2PC) protocols. Specifically, secure inference achieves the following: If a user has private data x on which she wants to find the output of inference $M(w, x)$ on a model M with weights w, then this should be possible, without any trusted intermediary, and without the user learning anything about w (except for $M(w, x)$) or the model provider learning anything about x. Recent advancements in secure inference include (Mohassel and Zhang 2017; Juvekar, Vaikuntanathan, and Chandrakasan 2018; Mishra et al. 2020; Rathee et al. 2020; Huang et al. 2022).

With these advances, is secure inference for deep models practical today? To ground a notion of what is practical, we take a specific example of chest X-ray interpretation. The largest public dataset of labelled chest X-rays is CheXpert (Irvin et al. 2019) with over 200K X-rays of size 320x320 labelled with 14 labels of 3-classes. Deep models such as DenseNet-121 with 121 layers and over 7.5M parameters are required to achieve state-of-the-art accuracy on this task, *i.e.*, an AUC of around 0.888. We could consider secure inference practical if a such a model trained on CheXpert could be used for securely interpreting X-rays with negligible loss of accuracy and in a commercially viable manner. This imposes three requirements. First, there should be automated processes to translate a model, such as DenseNet-121 with several different operators and a very deep network, into an executable for secure inference. Second, the translation must result in no deviation between outputs of clear-text and secure inference, ensuring no loss in accuracy or robustness of the model. Third, the cost of secure inference as measured by communication and latency should be manageable.

Amongst all existing approaches to secure inference,

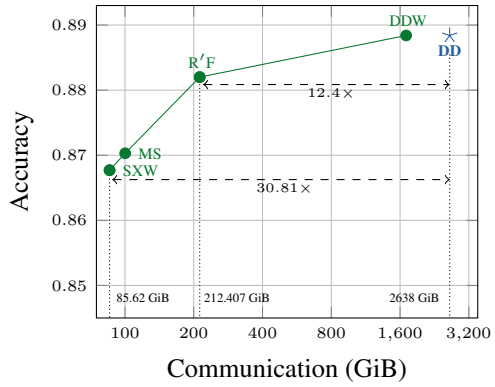

Figure 1: **Communication-accuracy Pareto Curve of Proposed models** (marked as •): We find a range of models that significantly improve on communication while being accurate relative to SOTA (marked as ⋆). Refer to Table 2 for the description of model mnemonics used here.

only the CrypTFlow2 framework satisfies the first two requirements[1]. The front-end of CrypTFlow2 can translate a TensorFlow/PyTorch model for secure inference with support for a wide range of common operators. Also, all secure inference protocols work with fixed-point ML models and CrypTFlow2 provides faithful truncation that ensures bit-wise equivalence between clear-text and secure inference, a highly desirable guarantee on the correctness of secure execution. CrypTFlow2 is a state-of-the-art system for secure inference and is the only one to have demonstrated accurate and secure inference at this scale (*i.e.* deep models over 320×320 size images) (Soin et al. 2021). Here, CrypTFlow2 provides highly efficient specialized cryptographic protocols. However, the cost of the resultant secure inference can be still impractically large: Almost 3TB of communication is required to interpret a single X-ray on DenseNet-121. This large overhead remains an outstanding challenge in making secure inference practical. In this paper, we address the high overheads of secure inference in CrypTFlow2 with three ways to optimize the ML model to be crypto-friendly.

1. Efficient Secure Inference of Linear Layers. We profiled the communication cost of individual layers of DenseNet-121 on CrypTFlow2 and found that linear layers accounted for atleast 96% of the communication cost (Table 1). This is in contrast to several other studies on efficient secure inference (Mishra et al. 2020; Ghodsi et al. 2020; Chaudhari, Jagielski, and Oprea 2022; Jha et al. 2021) which identified non-linear layers such as ReLU activation as the bottleneck. CrypTFlow2 provides novel specialized protocols for secure ReLU computation, that significantly reduces their cost, shifting the performance bottleneck to linear layers. Since most of the existing techniques on efficient clear-text inference (Howard et al. 2019; Ma et al. 2018) optimize FLOP-heavy linear layers, we directly adopt them for secure inference. In particular, we adopt optimized network backbones, factorized depth-wise separable convolution, and Shuffle operators. We show that communication cost can be significantly reduced ($1.5\times$ to over $8\times$) by reducing FLOPs, while achieving high accuracy.

2. A custom Crypto-friendly X-Operator. Going beyond existing techniques, we design a new custom crypto-friendly X-operator as a drop-in replacement for the *bottleneck cells* (He et al. 2016). The X-operator is composed of grouped point-wise convolution, followed by an addition on the outputs of "parallel" shuffle and depthwise convolution, followed by another grouped point-wise convolution. For the MobileNetV3-Large backbone, the X-operator reduces the communication cost by over 90% while incurring only a 0.011 drop in AUC (Table 2).

3. Winograd Algorithm for Convolutions. While the above two optimizations change the model by introducing efficient operators, our third contribution is a compiler technique whereby we rewrite convolutions to use the fast Winograd algorithm (Lavin and Gray 2016). This reduces the number of multiplications, which are expensive in secure inference, while increasing number of additions which do not incur any communication. We add support for the Winograd algorithm as part of the Athos (Kumar et al. 2020) compiler in CrypTFlow2 to automatically optimize any linear layer. As an example, for the DenseNet-121 network, the Winograd algorithm reduces communication cost by 55% without any loss in accuracy. We show similar results across other network backbones as well.

Our Results. In combination, these optimizations result in a range of models[2] panning a Pareto curve of accuracy and inference cost on the CheXpert dataset, as shown in Figure 1. For instance, we identify a model with about $12.4\times$ lower communication and a negligible drop in AUC of 0.006, and another with over $30\times$ lower communication (bringing the communication below 100 GB and latency below 90s) with a drop in AUC of 0.02. In contrast, the architecture optimizations proposed in prior work (Mishra et al. 2020; Ghodsi et al. 2020; Chaudhari, Jagielski, and Oprea 2022; Jha et al. 2021) can reduce communication cost by only 1% to 4% for CheXpert. We also note that the problems of efficient clear-text inference and efficient secure inference are distinct as the optimizations we consider don't offer commensurate benefits for clear-text inference. For instance, our model that lowers secure inference cost by $30\times$, improves clear-text latency on an Intel Xeon CPU by only $7\times$ and a V100 GPU by only $4\times$. Finally, our significant cost reductions for secure inference demonstrate that crypto-friendly ML models can be designed to make secure inference practical for realistic models.

Other Related Works. There are two primary approaches to improve the performance of secure inference on a given task: changing cryptography or changing the model. These two approaches are complementary. In the latter category, (Mishra et al. 2020; Ghodsi et al. 2020; Chaudhari, Jagielski, and Oprea 2022; Jha et al. 2021) reduce the number of ReLUs. Cryptonets (Gilad-Bachrach et al. 2016) and ngraph-

[1]We discuss other frameworks in the background section.

[2]We consider only non-ensemble models in this work. We expect the benefits to translate for ensembles.

HE (Boemer et al. 2019) replace ReLUs entirely by quadratics. CoiNN (Hussain et al. 2021) and SiRNN (Rathee et al. 2021) use varying bitwidths to balance accuracy and efficiency. Reducing bitwidths is complementary to our work and improves the efficiency of secure evaluation of both linear and non-linear layers.

Background

Here, we provide the necessary background on secure 2-party computation (2PC) and the cost profile of different operations with the state-of-the-art protocols in the CrypTFlow2 framework that guide our operator choices in this paper for secure inference.

2PC and Secure Inference. 2PC (Yao 1986; Goldreich, Micali, and Wigderson 1987) allows two parties P_0 and P_1 holding private inputs x and y, respectively, to *securely* compute $f(x, y)$ for any public function f. It provides a formal security guarantee that a party does not learn anything about the other party's private input beyond what can be deduced from the function output. It is easy to see that secure inference can be solved using 2PC between the model owner, say P_0, holding the model weights w and the client, say P_1, holding the data point x who want to compute $M(w, x)$, where M is the public model architecture.

All works on 2-party secure inference use cryptographic primitive called *additive secret sharing* that allows to split private data (*secret*) into 2-parts called *secret shares* such that a single share completely hides the secret and the secret shares can be added to reveal the secret. Parties start by secret sharing their inputs, and then for each layer of the network, parties start with secret shares of the input to that layer and run a protocol to securely compute the secret shares of the output of that layer. All works on secure inference provide such protocols for both linear and non-linear layers. These protocols require communicating bits that are indistinguishable from random, which makes it secure.

Cost characteristics of secure inference protocols. 2PC protocols are interactive *i.e.,* both the parties exchange cryptographic messages over multiple rounds of interaction. And for secure inference of large models, the cost of the secure inference is governed by the communication of the protocol, *i.e.*, the total amount of data being exchanged between the two parties. It is important to note that cost of different operators in 2PC is quite different from corresponding cost over the clear-text (that is, when the inputs are known in the clear for computation). For instance, while additions and multiplications cost roughly the same over clear-text, in 2PC additions are free and multiplications are expensive. This is because the secret sharing scheme is additive in nature and *no* interaction is needed for adding two secret values. Similarly, any operation that permutes the data in a tensor requires no interaction since both parties can individually permute their shares of the tensor, e.g., shuffle operator.

On the other hand, non-linear operations such as ReLU and Maxpool require interaction. Works prior to CrypTFlow2 such as MiniONN (Liu et al. 2017), Gazelle (Juvekar, Vaikuntanathan, and Chandrakasan 2018), Delphi (Mishra et al. 2020) and others used garbled circuits (Yao 1986) for these functions that were communication intensive. CrypTFlow2 provides novel specialized 2PC protocols for non-linear operations that reduce their cost by more than $10\times$.

ML to 2PC. Directly working with 2PC protocols requires expertise in cryptography. Hence, in recent years, several frameworks have been developed that automatically generate secure inference implementations from the clear-text ML code in various threat models (Kumar et al. 2020; Dalskov, Escudero, and Keller 2020; Dahl et al. 2018; Ryffel et al. 2018; Knott et al. 2021). The users train ML models in Keras, PyTorch, or Tensorflow and use these frameworks to get secure inference implementations at the push of a button. Among these, only CrypTFlow2 generates 2PC protocols; the other frameworks require additional trust assumptions. Furthermore, CrypTFlow2 guarantees correctness, *i.e.*, the the clear-text execution is bitwise equivalent to the generated 2PC implementation. This guarantee is critical for the feasibility of our evaluation. Since 2PC protocols are expensive, it is intractable to run 2PC implementation for evaluating accuracy on various backbones and optimized networks. Armed with this correctness guarantees, we use the clear-text implementations to measure accuracy soundly.

Characterising Secure Inference of DNNs

Deep models for computer vision, such as DenseNet-121, ResNet-50 (He et al. 2016), MobileNetV3-Large (Howard et al. 2019), and ShuffleNetV2 (Ma et al. 2018) contain a large number of layers and various operators, which may be classified as either linear or non-linear. Linear layers consist of operations such as convolutions, matrix multiplications, batch normalization and use additions/multiplications. Non-linear layers consist of activation functions such as ReLU, pooling layers such as max-pool, and classifier layers such as argmax. We profile the communication cost of secure inference for these common deep models, and identify linear layers (particularly convolutions) to be the bottleneck.

We profile the four models with an input image size of 320×320 using CrypTFlow2, instrumented to record communication cost at an operator level. We report the split of communication cost for linear and non-linear layers in Table 1. We find that, across the four models, at least 96% of the communication is attributed to linear layers[3]. This dominance of the cost of linear layers in secure inference differs from prior work which found non-linear layers to be more expensive (Mishra et al. 2020; Ghodsi et al. 2020; Jha et al. 2021). The reasons for this difference are two-fold: 1) Prior work primarily focused on models for small images such as in MNIST and CIFAR datasets (up to 32x32) wherein the linear layers are less computationally expensive. In contrast, we work with realistic images such as X-ray images of size 320x320. 2) Prior work used secure inference protocols with Garbled Circuits (GC) (Yao 1986) which are communication intensive for non-linear layers (Mishra et al. 2020). In

[3]Although these costs are obtained using oblivious transfer based 2PC protocol in CrypTFlow2, we believe that our techniques should give similar benefits when executed with SOTA HE-based protocols (Huang et al. 2022) where bottlenecks are similar.

Network	% of linear			% of non-linear
	% of convolution	% of other linear	% all	
DenseNet-121	97.11%	1.58%	98.69%	1.31%
ResNet-50	96.67%	1.89%	98.56%	1.44%
MobileNet-V3	90.56%	5.72%	96.28%	3.72%
ShuffleNet-V2	92.82%	4.10%	96.92%	3.08%

Table 1: **Split of communication across linear and non-linear layers.** Across DNNs, linear layers dominate communication. Amongst them, convolution layers dominate.

contrast, the current SOTA secure inference methods such as CrypTFlow2 provide specialized protocols for non-linear layers that are over $10\times$ cheaper than GC.

Within linear layers, we examine the communication cost between convolution layers and other layers (including batch normalization). We find that convolutions account for at least 90% of the communication cost in all our computer vision models (Table 1). Again, this is in contrast to equivalent numbers for clear-text inference: In MobileNetV3-Large for instance, convolutions account for about 70% of the computational cycles when running on a CPU.

The above profiling results indicate opportunities for making secure inference efficient. In contrast to prior work with other protocols, CrypTFlow2's communication cost is localized within linear layers. And in contrast to clear-text inference, within linear layers, convolutions are most expensive in CryptFlow2. Furthermore, tensor additions are local operations and thus communication-free, while multiplications are expensive and require 1.2KB of communication per operation. Thus, **reducing the number of multiplication operations**, denoted #MULT, emerges as an effective strategy for efficient secure inference.

Efficient Alternatives to Convolution

Designing deep models for efficient clear-text inference has been a widely studied problem (Sze et al. 2017). Such attempts have to consider several aspects such as reducing floating-point operations (FLOPs), optimizing for a given architecture, exploiting locality in memory access patterns, and so on. However, the above profiling of communication cost for secure inference suggests a simple strategy of reducing #MULTs, which are situated primarily in convolution operations. Consequently we consider two existing approaches to optimize convolution operations: (a) factorized convolution, and (b) multiplication-free shuffle operation. We also demonstrate out-sized reductions in cost for secure inference compared to clear-text with these optimizations.

Factorized Convolution. Convolutions form a key component in models for computer vision, and are designed to aggregate information across spatial (usually two: x- and y-axes) and depth or channel (usually many) axes. Like tensor multiplication, convolution can also be factorized, along spatial axes (Gholami et al. 2018), depth axis (Chollet 2017; Howard et al. 2019), or both (Ganesan and Kumar 2021; Selvam, Ganesan, and Kumar 2021). Indeed, most efficient models (that are deployed in low-resource devices such as mobile phones) use depthwise separable convolutions (DS-Convolution), as shown in Figure 2(b). The aggregation of

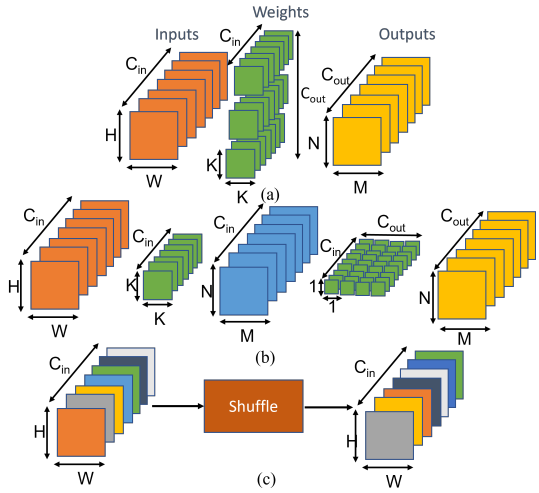

Figure 2: **Operators used in clear-text.** (a) Dense Conv, (b) Factorized depthwise-separable Conv, and (c) Shuffle

information along spatial and depth axes are separated out: Independent 2-D convolutions on each input depth are followed by pointwise or 1×1 convolutions.

Depthwise factorization can significantly reduce #MULT. For instance, a standard convolution with a filter of size $C_{\text{out}} \times C_{\text{in}} \times K \times K$ on an input of size $C_{\text{in}} \times H \times W$ and output of size $C_{\text{out}} \times N \times M$ requires $NMC_{\text{out}}K^2C_{\text{in}}$ #MULT. If this was converted into a DS-Convolution, it would require $NMC_{\text{in}}(K^2 + C_{\text{out}})$ #MULT, a reduction by a factor of $\frac{K^2C_{\text{out}}}{K^2+C_{\text{out}}}$. As an example, for a filter with a spatial size of $K = 3$ and input features of size $C_{\text{out}} = 32$, moving from dense to DS-convolution can reduce #MULT by over $7\times$.

Do the reduction in #MULT with factorized convolution translate to lower communication costs for secure inference? We empirically evaluate this with a convolution operator with $K = 3, C_{\text{in}} = 3, C_{\text{out}} = 64$, and various input sizes. We compute the communication with CrypTFlow2 for both dense and DS-convolution while varying the image size from 16×16 to 512×512. The results, shown in Figure 3, indicate a consistent $5\times$ reduction in communication cost for DS-convolution across image sizes. However, this reduction in communication cost needs to be traded-off with any drop in accuracy. We evaluate this for models on the CheXpert dataset and show that the accuracy loss is negligible.

Shuffle Operation. In channel-wise Shuffle (Ma et al. 2018), channels of a feature map are randomly permuted, enabling information flow across channels (see Figure 2 (c)). When implemented for clear-text inference, permuting channels has memory and thus latency overheads. However, for 2PC, Shuffle does not add #MULT and is a local operation that is communication-free. Thus, for secure inference, we can reduce the number of #MULT by sharing the compute available between convolution and shuffle. Recall that DS-convolution has C_{in} independent 2-D depthwise convolutions followed by channel aggregating 1×1 pointwise convolution. When using depthwise convolutions

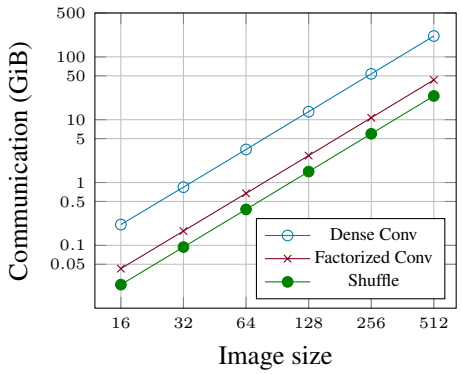

Figure 3: **Scaling of efficient clear-text operators.** We observe consistent communication benefits with factorized convolution and shuffle for increasing image sizes

in composition with Shuffle, $C_{\text{in}}/2$ channels are randomly permuted while the other $C_{\text{in}}/2$ channels have independent 2-D convolutions reducing #MULT by half with respect to DS-convolution.

Just as with DS-convolution, we empirically compute the communication cost of the Shuffle operator for various input image sizes. We see that moving from DS-Convolution to Shuffle reduces the communication cost by $1.8\times$, a reduction that is consistent across image sizes. (see Figure 3)

Gains in Clear-text vs Secure Inference. We showed that DS-convolution reduced communication costs by $5\times$ relative to dense convolution, and Shuffle reduced it by a further $1.8\times$ for an aggregate saving of $9\times$. For clear-text inference, DS-convolution reduces the average execution time on a CPU only by $1.4\times$ and GPU by only $2\times$ relative to dense convolution, with shuffle not improving the execution time any further. These out-sized gains in secure inference suggest a direct and sensitive dependence on #MULT while efficient clear-text inference depends on multiple factors including the architecture on which it is deployed.

A Custom Efficient Operator for 2PC

In the previous section, we showed that reducing #MULT is an effective strategy to reduce communication cost for secure inference, and that existing efficient operators for clear-text inference provide out-sized gains. Next, we design a custom operator specifically suitable for secure inference.

Factorized Point-Wise Convolution. Current deep models for computer vision use a cell-based design, i.e., a basic building-block is identified and then replicated with different dimensions in various layers to build a deep model. For instance, bottleneck cells have been commonly used since they were shown to be effective initially for ResNet-50 (He et al. 2016). A bottleneck cell contains a $K \times K$ dense convolution sandwiched between two point-wise convolutions (Figure 4 (a)). As discussed earlier, efforts have been made to make dense convolution more efficient by using factorization or Shuffle operators. Point-wise convolutions are expressible as matrix multiplications and have not received much attention for efficiency as they do not dominate clear-text inference costs.

To understand the corresponding costs for secure inference, we profiled a bottleneck cell (see Figure 4 (a)) with dense, DS-convolution and Shuffle for a feature map of size $3\times320\times320$, a dense filter of size 3×3 and $C_{\text{in}} = 3, C_{\text{out}} = 64$ with intermediate channels $C' = 48$. The fraction of the communication cost attributed to point-wise convolution for the three cases are 13.4%, 89.3%, and 95%. Thus in these optimized operators, the performance bottleneck has shifted to the point-wise convolution.

To address this, we propose to novel factorized point-wise convolution for 2PC. Consider a point-wise convolution of an input of size $C_{\text{in}} \times H \times W$ with a filter of size $C_{\text{out}} \times C_{\text{in}} \times 1\times1$. This entails HW tensor dot-products of tensors of size C_{in} for every C_{out} filter, a total of $C_{\text{out}}C_{\text{in}}HW$ #MULT. We propose to factorize this into two groups, i.e., divide the input map into two tensors with $C_{\text{in}}/2$ channels each and perform two independent point-wise convolution with two filters of size $C_{\text{in}}/2$, followed by a tensor composition. With this, we obtain 2 outputs per C_{in} multiplications. Hence, to maintain the same output feature map size, we reduce the number of filters to $C_{\text{out}}/2$, effectively cutting the overall #MULT by half to $C_{\text{out}}C_{\text{in}}HW/2$.

"Parallel" composition of Shuffle and Depth-wise convolution. As discussed earlier, in secure inference both shuffle operations and additions of tensors are communication-free. Thus, they can be used to express information flow across channels and aggregate information respectively without any cost. Recall that depthwise convolutions perform independent 2-D convolutions on each channel. We propose to augment depthwise convolution by adding a "parallel" path where shuffle operation is also performed on *all* channels enabling information flow across channels. Then the results of the shuffle and convolutions paths are aggregated with a tensor addition, which does not incur communication. Such parallel composition is expected to double the latency cost of clear-text inference on CPUs, but is uniquely efficient for 2PC.

X-operator. We combine the above two propositions into a single operator that we call the X-operator, as visualized in Figure 4. It comprises of a parallel shuffle and depthwise convolution, that is sandwiched between two point-wise convolutions that are factorized into two groups. The X-operator can be a drop-in replacement for a bottleneck cell. For instance, in layer 1 of ResNet-50, the bottleneck cell uses dense convolution and has 1.46B #MULT. Replacing the dense convolution within the bottleneck with DS-Convolution and Shuffle, the #MULT is reduced to 0.53B ($2.75\times$). Instead, replacing the bottleneck layer with the X-operator reduces #MULT to 0.27B ($5.3\times$).

X-operator for Secure Inference. We empirically compute the communication cost of X-operator in secure inference with parameters $C_{\text{in}} = 16$, $C_{\text{out}} = 64$ with $K = 3$. The input and output channels for the sandwiched bottleneck $C' = 48$. We observe that X-operator gives a reduction in communication cost of $1.5\times$ relative to bottleneck cell with half DS-convolution and half Shuffle, and $7.59\times$ relative to bottleneck cell with dense convolution, consistently

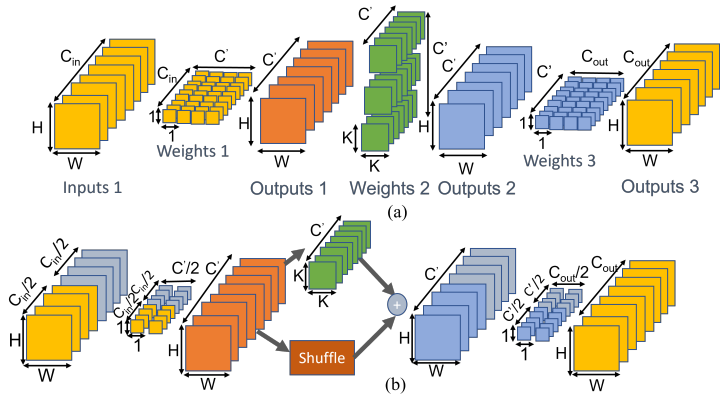

Figure 4: (a) Bottleneck cell, (b) X-operator. The proposed operator optimises both pointwise and dense convolutions with factorisation and parallel composition.

across image sizes. This establishes the effectiveness of designing custom operators that are crypto-friendly.

Here, we note the value of an automated compiler flow. We designed a custom-operator network that can be trained on a large dataset, and were able to compile it into a secure executable through Athos. This automated flow is crucial to accelerate the discovery of potentially many other crypto-friendly operators.

Winograd Algorithm for Convolutions

In the previous two sections, we proposed ways to reduce the cost of secure inference with existing and custom operators. The adoption of these operators is conditioned on the accuracy of models using them. In this section, we propose an alternative algorithm to implement convolution operator while retaining functional equivalence.

Given the dominance of convolution operation in deep models, several methods exist to improve its performance. In particular, the Winograd algorithm (Lavin and Gray 2016) implements convolution with fewer multiplications at the cost of increased number of additions and storage of intermediate outputs. In clear-text inference, Winograd algorithm has not been shown to reduce inference cost significantly, due to the relatively similar costs of addition and multiplication and also the cost of additional memory accesses (Madisetti 1997; Lavin and Gray 2016). However, for secure inference Winograd algorithm can effectively reduce communication cost given that it reduces #MULT.

Winograd Algorithm in Secure Inference. Consider the convolution of an image I of size $N \times N$ with a filter F of size $K \times K$ to obtain an output O of size $M \times M$. [4] In dense convolution this requires M^2K^2 #MULT. With the Winograd algorithm, the same operation can be computed with $(M + K - 1)^2$ #MULT. As an example, for output image of size 5×5 and filter of size 3×3, the reduction in #MULT is about $4.5\times$. Formally, the algorithm computes:

$$O = A^T[(B^T IB) \odot (GFG^T)]A$$

[4]Note that the relation $N = M + K - 1$ holds

where (A, B, G) are constant and public tensors dependent only on the image/filter sizes and can be pre-computed. $\odot$ is element-wise multiplication. For secure inference, the terms $B^T IB$ and GFG^T can be computed locally due to linearity of secret sharing. Evaluating $\odot$ requires $(M + K - 1)^2$ element-wise multiplications, which incurs 2PC communication cost. The final $A^T[\cdot]A$ can also be computed locally without any communication cost.

Tiling. Using the Winograd algorithm for convolving a large image in a single shot causes numerical errors because of the presence of irreducible fractions in the (A, B, G) matrices mentioned in the main text. The #MULT for convolving an $M \times M$ image with a $K \times K$ using Winograd without tiling

$$\beta_1(M, K) = (M + K - 1)^2$$

To convolve a large image we split it into multiple equal sized tiles and perform Winograd convolution for each input tile. Subsequently, the outputs tiles are then pieced together to obtain the complete image. With tiles of size $n \times n$, the number of tiles required to cover the image along an axis is

$$T = \left\lceil \frac{M - K + 1}{n - K + 1} \right\rceil$$

Convolving each tile invokes n^2 multiplications. Thus, #MULT for the tiled Winograd algorithm is

$$\beta_2(M, K; n) = T^2 n^2$$

It is straightforward to show that for all image sizes $M \geq n$, the multiplications invoked without tiling is lesser than with tiling *i.e.,*

$$\beta_1(M, K) \leq \beta_2(M, K; n)$$

For instance, for a 320×320 image 3×3 filter, #MULT for standard convolution is 910,116 and non-tiled Winograd algorithm is 102,400. On the other hand, the tiled Winograd algorithm with tile size 6×6 would use 230,400 multiplications.

Details of Tiling. Images with arbitrary size $M \times M$ need not be divisible neatly into tiles of size $n \times n$. The right/bottom edges of the image may need to be covered by tiles of a smaller size $n' \times n'$. Here $r \leq n' \leq n-1$. For ease of implementation, we pad these edges of the image with 0 so that a *single* tile size $n \times n$ can evenly cover the image. However, padding incurs an increase in the number of multiplications by a factor γ, which we estimate

$$\gamma = \frac{\text{\#multiplications with padding}}{\text{\#multiplications without padding}}$$

$$= \frac{T^2 n^2}{[(T-1)n + n']^2} = \left[\frac{1}{1 - (\frac{1 - n'/n}{T})}\right]^2$$

This factor is maximized when $n' = K$. Under a first-order approximation, we have

$$\gamma \approx 1 + 2\left(\frac{1 - K/n}{T}\right)$$

Convolving large images entails $T >> 1$ and hence the overhead factor γ due to padding is negligible.

Compiler support. In the Athos compiler front-end, we enable an ML developer to specify whether a convolution operation is to be implemented as a Winograd algorithm. This automated feature enables a functionally equivalent implementation while reducing the 2PC communication cost.

Limitations. Winograd transformation can only be applied to convolutions with a square $K \times K$ filter with $K > 1$. Therefore if the network is dominated by pointwise convolutions - the gains diminish. Moreover, the nature of the Winograd transformation is only applicable to convolutions with stride $s = 1$. Making the Winograd transformation work for stride $s > 1$ is an open research problem with multiple solutions having been proposed (Pan and Chen 2021; Huang et al. 2021; Yepez and Ko 2020) in the recent past. Using Winograd convolution for $r = 7$ causes a lack of numerical precision, and hence we avoid it. Due to the above technicalities, while this transformation may not always be applicable, it still gives promising gains when applied to networks with large dense convolution layers.

Experimental Evaluation

In this section, we share the methodology and results from the experiments evaluating our proposed optimizations.

Methodology

Models. We evaluate the optimizations with five different models that we have discussed in the paper thus far: DenseNet-121, ResNet-50, ResNet-18, MobileNetV3-Large, and ShuffleNetV2. We consider different variants of these models, starting with a baseline variant where all convolution operations are dense convolutions. Then, we progressively use the optimized operators: DS-Convolution, shuffle operator, and X-operator. Most of these models (except ResNet-18) use *bottleneck cell* as the basic building block, *i.e.* a dense convolution sandwiched between two pointwise convolutions. As we *factorize* the dense convolution into DS-Convolution, one can observe two back to back 1×1 pointwise convolutions, one arising from the factorization while the other existing within *bottleneck cell*. We combine these two repeating pointwise convolutions for parameter efficiency. For the most efficient model family, ShuffleNetV2, we optimize each variant with the Winograd algorithm. For all other models, Winograd is applied only on the dense variant where it is found to be most effective.

Training setup. We use PyTorch (Paszke et al. 2019) to train the models on the CheXpert (Irvin et al. 2019) challenge training dataset consisting of 224,316 chest radiographs from 65,240 patients and evaluate them on the validation dataset from 200 patients involving 5 output classes : (a) Atelectasis, (b) Cardiomegaly, (c) Consolidation, (d) Edema, and (e) Pleural Effusion. The training labels of the dataset can be 0, 1, or uncertain (u). In practice, the u labels are converted to a 1 (U-Ones), 0 (U-Zeros), or ignored (U-Ignore). We convert the uncertain labels to 1 and utilise the U-Ones model throughout for training. We apply standard pre-processing for CheXpert with a center-crop of 320×320 and normalise the pixels with a mean of 0.533 and standard deviation 0.0349. We train model variants to maximise the area under the receiver operating curve (AUC) and report that as the accuracy metric. We use SGD as the optimizer with an initial learning rate of 0.001. We use FP32 precision and train the models for 10 epochs on a V100 cluster with a batch size of 16.

Secure inference. Models trained in PyTorch are exported into ONNX (Microsoft 2017) which is then processed by the Athos toolchain to output fixed-point models with bit-width 60 and scale 23. These parameters ensure that the accuracy of fixed-point models matches that of PyTorch. The cleartext fixed-point models are linked with the secure protocols in CrypTFlow2. We measure latency on a LAN setup where two VMs are connected by a network with 16Gbps bandwidth and 0.7 ms round-trip time. The two VMs have the following specification.

Processor	AMD EPYC 7763 64-Core
Clock Speed	2.44 Ghz
RAM	128 GB
Bandwidth	16 Gbps
Ping Time	0.7 ms

Results

In Table 2 we report the accuracy on CheXpert and relative gains in communication and latency costs with respect to DenseNet-121 for inference of a single input image on each of the model variants. We make the following observations:
Choice of model. For the Dense variants, the choice of the model itself affects the cost - ResNet-50 incurs more communication than DenseNet-121 while MobileNetV3-Large leads to a $2.18\times$ reduction for a reduction in AUC by 0.006.
Factorized DS-convolution. Factorizing dense convolution to DS-convolutions reduces communication cost by an average of $4.07\times$ across models. For instance, ResNet18 has $7.7\times$ lesser communication (R′F vs R′D) with very negligible loss in AUC. Gains are however not uniform across models — DenseNet-121 and ResNet-50 improve modestly.
Shuffle Operator. Using the Shuffle operator which optimizes DS-convolution further by using shuffle for half the channels leads to a marginal reduction in communication cost, by an average of $1.1\times$ w.r.t. to the corresponding model with factorized convolution. Specifically, MS has about $1.41\times$ lower communication cost than MF with a drop in AUC of 0.003. Models which had lower gains with factorized convolution also have lower gains with Shuffle.
X-Operator. Using the X-operator provides the largest gains for all models by an average of $8.88\times$ w.r.t. to corresponding models with dense convolutions. Specifically, SX has a $30\times$ lower communication cost than the baseline DenseNet-121 model at the cost of an AUC reduction of 0.02.
Winograd algorithm. When applied on dense convolution, Winograd reduces communication cost by an average of $2.12\times$ w.r.t. dense models without any reduction in accuracy. For DenseNet-121, this provides a competitive alternative to efficient operators which marginally reduce AUC.
Composition of Winograd with efficient operators. Wino-

Backbone	Operator	Winograd	Mnemonic	Communication Reduction	Latency Reduction	Average AuC on CheXpert
DenseNet-121	Dense		DD	1×	1×	0.888
	Dense	✓	DDW	1.55×	1.06×	0.888
	Factorized		DF	1.56×	1.06×	0.885
	Shuffle		DS	1.57×	1.06×	0.886
	X-op		DX	2.37×	1.33×	0.880
ResNet-50	Dense		RD	0.71×	0.76×	0.884
	Dense	✓	RDW	0.96×	0.87×	0.882
	Factorized		RF	1.18×	1.08×	0.883
	Shuffle		RS	1.18×	1.17×	0.882
	X-op		RX	2.10×	1.43×	0.876
ResNet-18	Dense		R′D	1.61×	1.90×	0.884
	Dense	✓	R′DW	3.61×	3.31×	0.884
	Factorized		R′F	12.41×	7.27×	0.883
	Shuffle		R′S	12.83×	8.25×	0.882
	X-op		R′X	19.10×	8.77×	0.866
MobileNetV3 Large	Dense		MD	2.18×	2.33×	0.882
	Dense	✓	MDW	8.30×	2.91×	0.882
	Factorized		MF	12.93×	9.11×	0.873
	Shuffle		MS	18.30×	9.11×	0.870
	X-op		MX	19.39×	9.91×	0.871
ShuffleNetV2	Dense		SD	5.50×	2.11×	0.879
	Dense	✓	SDW	9.07×	7.30×	0.879
	Factorized		SF	19.39×	12.84×	0.872
	Factorized	✓	SFW	19.66×	10.30×	0.872
	Shuffle		SS	19.67×	12.84×	0.872
	Shuffle	✓	SSW	19.67×	12.84×	0.875
	X-op		SX	30.15×	15.35×	0.868
	X-op	✓	SXW	30.80×	17.09×	0.868

Table 2: Evaluating the benefits of models supporting different operators. The gains are relative to DenseNet-121 (DD) with 2.63TB of communication and 1524.9s latency.

grad algorithm can be applied to all model variants, as shown for the ShuffleNetV2 backbone. On the dense variant, the algorithm reduces communication by $1.7\times$ (SDW vs SD). However, for the three variants with efficient operators, the algorithm provides smaller gains between $1 - 2\%$.

Latency. For each variant, we also report the reduction in latency for a 2VM setup w.r.t. the baseline model. We note that in secure 2-party computation, communication gains can be obtained at higher latency costs. However, with the optimizations proposed, gains in both metrics are observed.

Prior works. As discussed, prior work has focused on optimizing non-linear layers which are not the bottleneck in CrypTFlow2. Thus application of optimizations from prior work will only yield negligible reduction in communication costs relative to the large reduction we demonstrate through choices of models, operators, and algorithms. For instance, even if we were to remove all ReLUs, as studied in (Mishra et al. 2020; Ghodsi et al. 2020; Chaudhari, Jagielski, and Oprea 2022; Jha et al. 2021), from DenseNet-121 (DD), the reduction in communication cost is only 0.8%.

Conclusion

We argue for designing models that have low 2PC costs to make secure inference practical. To address this we propose combining (a) optimizations on the model such as using different network backbones, efficient clear-text operators, custom crypto-friendly operators, and fast convolution algorithms, with (b) a secure inference framework like CrypTFlow2 which provides compilation of arbitrary deep models into functionally equivalent secure inference. In the chest X-ray interpretation task, we show that the optimizations lead to large reductions in secure inference costs with small drops in accuracy.

In the future, we plan to apply orthogonal optimizations commonly used in secure inference such as varying bitwidths (Hussain et al. 2021; Rathee et al. 2021), reducing communication by increasing computation (Soin et al. 2021; Huang et al. 2022; Boyle et al. 2019; Yang et al. 2020), and hardware acceleration (Watson, Wagh, and Popa 2022). We believe more gains remain to be unlocked at the intersection of design of secure inference frameworks and efficient models.

References

Boemer, F.; Lao, Y.; Cammarota, R.; and Wierzynski, C. 2019. nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data. In *CF*.

Boyle, E.; Couteau, G.; Gilboa, N.; Ishai, Y.; Kohl, L.; Rindal, P.; and Scholl, P. 2019. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In *CCS*, 291–308. ACM.

Chaudhari, H.; Jagielski, M.; and Oprea, A. 2022. SafeNet: Mitigating Data Poisoning Attacks on Private Machine Learning. Cryptology ePrint Archive, Paper 2022/663. https://eprint.iacr.org/2022/663.

Chollet, F. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1800–1807.

Dahl, M.; Mancuso, J.; Dupis, Y.; Decoste, B.; Giraud, M.; Livingstone, I.; Patriquin, J.; and Uhma, G. 2018. Private Machine Learning in TensorFlow using Secure Computation. *CoRR*.

Dalskov, A. P. K.; Escudero, D.; and Keller, M. 2020. Secure Evaluation of Quantized Neural Networks. *PoPETs*.

Ganesan, V.; and Kumar, P. 2021. Design and Scaffolded Training of an Efficient DNN Operator for Computer Vision on the Edge. *ACM Transactions on Embedded Computing Systems (TECS)*.

Ghodsi, Z.; Veldanda, A. K.; Reagen, B.; and Garg, S. 2020. CryptoNAS: Private Inference on a ReLU Budget. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *NeurIPS*.

Gholami, A.; Kwon, K.; Wu, B.; Tai, Z.; Yue, X.; Jin, P.; Zhao, S.; and Keutzer, K. 2018. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 1638–1647.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K. E.; Naehrig, M.; and Wernsing, J. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 201–210.

Goldreich, O.; Micali, S.; and Wigderson, A. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, 218–229. New York, NY, USA: Association for Computing Machinery. ISBN 0897912217.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

HIPAA. 2012. Guidance Regarding Methods for De-identification of Protected Health Information in Accordance with the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule. https://www.hhs.gov/hipaa/for-professionals/privacy/special-topics/de-identification/index.html.

Howard, A.; Sandler, M.; Chen, B.; Wang, W.; Chen, L.; Tan, M.; Chu, G.; Vasudevan, V.; Zhu, Y.; Pang, R.; Adam, H.; and Le, Q. 2019. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 1314–1324. Los Alamitos, CA, USA: IEEE Computer Society.

Huang, C.; Dong, X.; Li, Z.; Song, T.; Liu, Z.; and Dong, L. 2021. Efficient Stride 2 Winograd Convolution Method Using Unified Transformation Matrices on FPGA. In *2021 International Conference on Field-Programmable Technology (ICFPT)*, 1–9.

Huang, Z.; jie Lu, W.; Hong, C.; and Ding, J. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *31st USENIX Security Symposium (USENIX Security 22)*, 809–826. Boston, MA: USENIX Association. ISBN 978-1-939133-31-1.

Hussain, S. U.; Javaheripi, M.; Samragh, M.; and Koushanfar, F. 2021. COINN: Crypto/ML Codesign for Oblivious Inference via Neural Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, 3266–3281.

Irvin, J.; Rajpurkar, P.; Ko, M.; Yu, Y.; Ciurea-Ilcus, S.; Chute, C.; Marklund, H.; Haghgoo, B.; Ball, R. L.; Shpanskaya, K. S.; Seekins, J.; Mong, D. A.; Halabi, S. S.; Sandberg, J. K.; Jones, R.; Larson, D. B.; Langlotz, C. P.; Patel, B. N.; Lungren, M. P.; and Ng, A. Y. 2019. CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 590–597.

Jha, N. K.; Ghodsi, Z.; Garg, S.; and Reagen, B. 2021. DeepReDuce: ReLU Reduction for Fast Private Inference.

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, 1651–1669. Baltimore, MD: USENIX Association. ISBN 978-1-939133-04-5.

Knott, B.; Venkataraman, S.; Hannun, A.; Sengupta, S.; Ibrahim, M.; and van der Maaten, L. 2021. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 4961–4973. Curran Associates, Inc.

Kumar, N.; Rathee, M.; Chandran, N.; Gupta, D.; Rastogi, A.; and Sharma, R. 2020. CrypTFlow: Secure TensorFlow Inference. In *IEEE Symposium on Security and Privacy*. IEEE.

Lavin, A.; and Gray, S. 2016. Fast Algorithms for Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4013–4021. Los Alamitos, CA, USA: IEEE Computer Society.

Liu, J.; Juuti, M.; Lu, Y.; and Asokan, N. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *CCS*.

Ma, N.; Zhang, X.; Zheng, H.-T.; and Sun, J. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Ferrari, V.; Hebert, M.; Sminchisescu, C.; and Weiss, Y., eds., *Computer Vision – ECCV 2018*, 122–138. Cham: Springer International Publishing. ISBN 978-3-030-01264-9.

Madisetti, V. 1997. *The digital signal processing handbook*. CRC press.

Microsoft. 2017. Open Neural Network Exchange. onnx.ai. Accessed: 2022-08-12.

Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; and Popa, R. A. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security Symposium*.

Mohassel, P.; and Zhang, Y. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, 19–38.

Pan, J.; and Chen, D. 2021. Accelerate Non-Unit Stride Convolutions with Winograd Algorithms. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ASPDAC '21, 358–364. New York, NY, USA: Association for Computing Machinery. ISBN 9781450379991.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, chapter 721, 12. Red Hook, NY, USA: Curran Associates Inc.

Rathee, D.; Rathee, M.; Goli, R. K. K.; Gupta, D.; Sharma, R.; Chandran, N.; and Rastogi, A. 2021. SIRNN: A math library for secure inference of RNNs. In *IEEE S&P 2020*.

Rathee, D.; Rathee, M.; Kumar, N.; Chandran, N.; Gupta, D.; Rastogi, A.; and Sharma, R. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS*.

Ryffel, T.; Trask, A.; Dahl, M.; Wagner, B.; Mancuso, J.; Rueckert, D.; and Passerat-Palmbach, J. 2018. A generic framework for privacy preserving deep learning. *CoRR*.

Selvam, S.; Ganesan, V.; and Kumar, P. 2021. Fuseconv: Fully separable convolutions for fast inference on systolic arrays. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 651–656. IEEE.

Soin, A.; Bhatu, P.; Takhar, R.; Chandran, N.; Gupta, D.; Alvarez-Valle, J.; Sharma, R.; Mahajan, V.; and Lungren, M. P. 2021. Production-level Open Source Privacy Preserving Inference in Medical Imaging. *CoRR*, abs/2107.10230.

Sze, V.; Chen, Y.-H.; Yang, T.-J.; and Emer, J. S. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12): 2295–2329.

Watson, J.-L.; Wagh, S.; and Popa, R. A. 2022. Piranha: A GPU Platform for Secure Computation.

Yang, K.; Weng, C.; Lan, X.; Zhang, J.; and Wang, X. 2020. Ferret: Fast Extension for Correlated OT with Small Communication. In *CCS*, 1607–1626. ACM.

Yao, A. C.-C. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 162–167.

Yepez, J.; and Ko, S.-B. 2020. Stride 2 1-D, 2-D, and 3-D Winograd for Convolutional Neural Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(4): 853–863.