A Related Works

Private inference has been a promising solution to protect both data and model privacy during deep learning inference. In recent years, there has been an increasing amount of literature on efficient private inference. According to the optimization technique, these works can be categorized into three types, i.e., 1) protocol optimization; 2) network optimization; and 3) joint optimization.

In protocol optimization, ABY [7] provides a highly efficient conversion between arithmetic sharing, boolean sharing and Yao's sharing, and construct mixed protocols. As an extension, ABY3 [34] switches back and forth between three secret sharing schemes using three-party computation (3PC). CypTFlow2 [39] proposes a new protocol for secure and comparison and division which enables effecient non-linear operations such as ReLU. SiRNN [38] further proposes 2PC protocols for bitwidth extension, mixed-precision linear and non-linear operations. CrypTen [27] proposes a software framework that provides a flexible machine learning focused API. More recently, SecFloat [37] proposes the crypto-friendly precise functionalities to build a library for 32-bit single-precision floating-point operations and math functions. These works lack consideration for neural network architecture and has limited communication reduction.

In network optimization, DeepReDuce [24] proposes to manually remove ReLUs with a three-step optimization pipline. SNL [6] proposes ReLU-aware optimization that leverages gradient-based NAS to selectively linearize a subset of ReLUs. CryptoNAS [18] uses ReLU budget as a proxy and leverages NAS to tailor ReLUs. PolyMPCNet[36] and SAFENet [33] replace ReLUs with MPC-friendly polynomial, while Sphynx [5] proposes an MPC-friendly ReLU-efficient micro-search space. SENet [30] innovatively measures the ReLU importance via layer pruning sensitivity and automatically optimize the network to meet the target ReLU budget. DeepReShape [24] finds that wider networks are more ReLU-efficient than the deeper ones and designs ReLU-efficient baseline networks with with FLOPs-ReLU-Accuracy balance. Network optimization mainly focuses on ReLU reduction which dominates the online communication, but total communication including convolution and truncation cannot be optimized.

Unluckily, only using either protocol or network optimization just leads to limited efficiency improvement. Delphi [44] jointly optimizes cryptographic protocols and network by gradually replacing ReLU with quadratic approximation. COINN [23] simultaneously optimizes quantized network and protocols with ciphertext-aware quantization and automated bitwidth configuration. Recently, [16] proposes to use Winograd convolution for reducing the number of multiplications and design the efficient convolution operation to reduce the communication cost. However, it does not take private inference into consideration for Winograd algorithm, and still suffers tremendous communication overhead. In this work, we jointly optimize the network and protocol and fully consider their coupling properties.

B Details of Experiment Setup

Private inference framework CoPriv adopts CypTFlow2 [39] protocol for private inference. We leverage the Athos [39] tool chain to convert both input and weight into fixed-point with the bit-width 41 and scale 12. We measure the communication and latency under a LAN setting [39] with 377 MBps bandwidth and 80ms echo latency. All of our experiments are evaluated on the Intel Xeon Gold 5220R CPU @ 2.20GHz.

Implementation of Winograd-based convolution protocol The convolution protocol with Winograd transformation and optimization is implemented in C++ with Eigen and Armadillo matrix calculation library [41] in the CrypTFlow2 [39] framework. We implement $F(2 \times 2, 3 \times 3)$ and $F(4 \times 4, 3 \times 3)$ transformation for convolution with stride of 1 and $F(2 \times 2, 3 \times 3)$ transformation when stride is 2. For CIFAR-100 dataset, we use $F(2 \times 2, 3 \times 3)$ transformation as the image resolution is small and for ImageNet dataset, we use $F(4 \times 4, 3 \times 3)$. We only apply $F(2 \times 2, 3 \times 3)$ for stride of 2 on ImageNet dataset. When evaluating CoPriv, we determine the optimal sender according to the analysis in Table 3 before inference. Winograd implementation enables us to measure the communication cost and latency of each convolution module.

Networks and datasets We apply our proposed CoPriv to the widely used lightweight mobile network MobileNetV2 [42] with different width multipliers, e.g., 0.75, 1.0 and 1.4 to trade off the model accuracy and efficiency. We evaluate the top-1 accuracy and online and total communication on both CIFAR-100 and ImageNet dataset.

Differentiable pruning and finetuning setups We first search and prune redundant ReLUs for 90 epochs and then finetune the pruned network for 180 epochs with SGD optimizer, cosine learning scheduler and 0.1 initial learning rate. We train our proposed CoPriv with self-distillation.

C Network Re-Parameterization Algorithm

Network/Structural re-parameterization is a useful technique proposed by RepVGG [13], and is extended to [10, 9, 12, 15, 11]. The core idea of re-parameterization is to decouple the training-time architecture (with high performance and low efficiency) and inference-time network architecture (with high efficiency). Re-parameterization is realized by converting one architecture to another via equivalently merging parameters together. Therefore, during inference time, the network architecture is not only efficient but also has the same high performance as the training-time architecture.

In this work, we can also leverage this technique to merge adjacent convolutions together after ReLU removal. For the network re-parameterization mentioned in Section 4.2, here we provide the following detailed algorithm 1 to equivalently merge the inverted residual block into a single dense convolution as shown in Figure 3. With the help of network re-parameterization, we further optimize the total communication including convolution and truncation.

Algorithm 1: Network Re-parameterization for Inverted Residual Block

Input : An inverted residual block with weights $W_{1\times1}$, $W_{3\times3}$, and $W'_{1\times1}$. The number of input and output channels N_{in}, N_{out}. The size of re-parameterized weights r.

Output : Regular convolution with re-parameterized weights W_r.

$W_r =$ `torch.eye(`N_{in}`);`
$W_r = W_r$`.unsqueeze(2).unsqueeze(2);`
$W_r =$ `torch.nn.functional.pad(`W_r, *pad*=`(`$\frac{r-1}{2}, \frac{r-1}{2}, \frac{r-1}{2}, \frac{r-1}{2}$`));`
$W_r =$ `torch.nn.functional.conv2d(`$W_r, W_{1\times1}$`);`
$W_r =$ `torch.nn.functional.conv2d(`$W_r, W_{3\times3}$, *padding*=$\frac{r-1}{2}$`);`
$W_r =$ `torch.nn.functional.conv2d(`$W_r, W'_{1\times1}$`);`
$W_{res} =$ `torch.zeros(`N_{out}, N_{in}, r, r`);`
for $i \in [0, \dots, N_{out}-1]$ **do**
  $W_{res}[i, i, \lfloor r/2 \rfloor, \lfloor r/2 \rfloor] = 1;$
$W_r = W_r + W_{res};$
return W_r;

D Details of Winograd Convolution

D.1 Comparison between Regular Convolution and Winograd Convolution

To help readers better understand the multiplication reduction of Winograd convolution, we demonstrate regular convolution and Winograd convolution in Figure 10. Given an input $I \in \mathbb{R}^{4\times4}$ and a filter $F \in \mathbb{R}^{3\times3}$, regular convolution requires $9 \times 4 = 36$ times multiplications (implemented using GEMM with im2col algorithm [4]) while $F(2 \times 2, 3 \times 3)$ Winograd transformation only requires $16 \times 1 = 16$ times multiplications (EWMM), which achieves 2.25× reduction. Moreover, $F(4 \times 4, 3 \times 3)$ with a larger tile size, i.e., 6 can further achieve 4× multiplication reduction. The improvement gets benefit from the Winograd's ability to convert im2col to EWMM and calculate the whole tile in Winograd domain at once.

D.2 Details of Input Tiling and Padding

Given a large 2D input $I \in \mathbb{R}^{l\times l}$, where $l > m + r - 1$, the core technique for ensuring the equivalence of regular convolution and Winograd convolution is input tiling and padding. The output size $l' = l - r + 1$, the input tile size $n = m + r - 1$ and the total tile number T per channel is

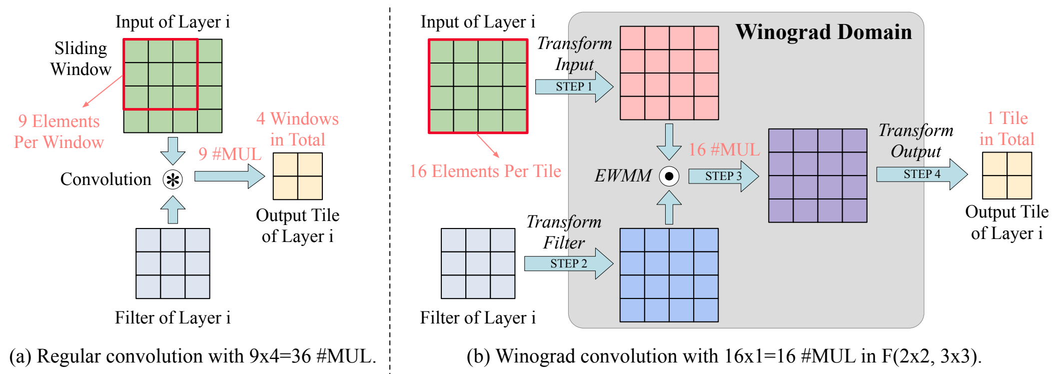

Figure 10: Comparison between (a) regular convolution and (b) Winograd convolution.

computed as

$$T = \lceil \frac{l'}{n} \rceil^2 = \lceil \frac{l-r+1}{m+r-1} \rceil^2,$$

where $\lceil \cdot \rceil$ denotes taking the upper bound value. For each tile, Winograd convolution is individually performed and results an output tile with $m \times m$ size. After all the tiles are computed with Winograd convolution, the output tiles are concatenated together to form the final output.

For some input size, the input cannot be covered by tiles. For instance, when leveraging $F(2\times2, 3\times3)$ on the input $I \in \mathbb{R}^{7\times7}$, the rightmost and bottom pixels cannot be divided into a complete tile. To solve this problem, we pad these positions with 0 to enable the tiles totally cover the whole input. The correctness and equivalence can be proved with Eq. 1. Also, [16] shows the overhead caused by padding is negligible.

D.3 Support for Stride of 2 Winograd Convolution

Conventional Winograd convolution only supports stride $s = 1$ convolution filter. However, in recent efficient neural networks, e.g., MobileNetV2, EfficientNet has several stride of 2 layers to reduce the feature map size by half. To enable extreme optimization for efficient networks, we introduce $F(2 \times 2, 3 \times 3)$ for stride of 2 Winograd convolution for private inference.

There are various methods to construct stride of 2 Winograd kernel such as dividing input and convolution filter into different groups [46]. However, it is not a simple way to implement stride of 2 Winograd kernel. [21] is an extremely convenient method using unified transformation matrices.

Based on [21], even positions of input and filter are computed by $F(2, 2)$ while odd positions are computed by regular convolution. Transformation matrices are derived as follows and can be computed using Eq. 1:

$$B^\top = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A^\top = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Correctness analysis. Here, we take a 1D algorithm as an example to prove the correctness Winograd convolution for stride of 2. The algorithm can be nested with itself to obtain a 2D algorithm [31].

Given input X and filter F as

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad F = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}, \quad Y = X \circledast F = \begin{bmatrix} z_0 \\ z_1 \end{bmatrix}.$$

First, we calculate regular convolution with stride of 2 using im2col algorithm [4] as

$$Y_1 = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_2 & x_3 & x_4 \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_0y_0 + x_1y_1 + x_2y_2 \\ x_2y_0 + x_3y_1 + x_4y_2 \end{bmatrix},$$

thus, $z_0 = x_0y_0 + x_1y_1 + x_2y_2$ and $z_1 = x_2y_0 + x_3y_1 + x_4y_2$.

Then, we calculate Winograd convolution for stride of 2 as

$$Y = A^\top \cdot [(GF) \odot (B^\top X)],$$

and then

$$Y_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \cdot [(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}) \odot (\begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix})],$$

and further simplify the calculation as

$$Y_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \cdot [(\begin{bmatrix} y_0 \\ y_1 \\ y_0 + y_2 \\ y_1 \\ y_2 \end{bmatrix}) \odot (\begin{bmatrix} x_0 - x_2 \\ x_1 \\ x_2 \\ x_3 \\ x_4 - x_2 \end{bmatrix})] = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0y_0 - x_2y_0 \\ x_1y_1 \\ x_2y_0 + x_2y_2 \\ x_3y_1 \\ x_4y_2 - x_2y_2 \end{bmatrix},$$

therefore, the convolution result is

$$Y_2 = \begin{bmatrix} x_0y_0 + x_1y_1 + x_2y_2 \\ x_2y_0 + x_3y_1 + x_4y_2 \end{bmatrix} = Y_1.$$

D.4 Transformation Matrices for Winograd Convolution

We provide the transformation matrices A, B, G for $F(2 \times 2, 3 \times 3)$ and $F(4 \times 4, 3 \times 3)$ Winograd transformation based on polynomial Chinese remainder theorem (CRT) or Lagrange interpolation [31].

For $F(2 \times 2, 3 \times 3)$, we have

$$B^\top = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}, \quad A^\top = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}.$$

For $F(4 \times 4, 3 \times 3)$, we have

$$B^\top = \begin{bmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} 1/4 & 0 & 0 \\ -1/6 & -1/6 & -1/6 \\ -1/6 & 1/6 & -1/6 \\ 1/24 & 1/12 & 1/6 \\ 1/24 & -1/12 & 1/6 \\ 0 & 0 & 1 \end{bmatrix},$$
$$A^\top = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{bmatrix}.$$

The correctness analysis is the same with Section D.3.

References

[1] Syed Asad Alam, Andrew Anderson, Barbara Barabasz, and David Gregg. Winograd convolution for deep neural networks: Efficient point selection. *ACM Transactions on Embedded Computing Systems*, 21(6):1–28, 2022.

[2] Barbara Barabasz, Andrew Anderson, Kirk M Soodhalter, and David Gregg. Error analysis and improving the accuracy of winograd convolution for deep neural networks. *ACM Transactions on Mathematical Software (TOMS)*, 46(4):1–33, 2020.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

[5] Minsu Cho, Zahra Ghodsi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Sphynx: A deep neural network design for private inference. *IEEE Security & Privacy*, 20(5):22–34, 2022.

[6] Minsu Cho, Ameya Joshi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Selective network linearization for efficient private inference. In *International Conference on Machine Learning*, pages 3947–3961. PMLR, 2022.

[7] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Repmlpnet: Hierarchical vision mlp with re-parameterized locality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 578–587, 2022.

[10] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1911–1920, 2019.

[11] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021.

[12] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10886–10895, 2021.

[13] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.

[14] Javier Fernandez-Marques, Paul Whatmough, Andrew Mundy, and Matthew Mattina. Searching for winograd-aware quantized networks. *Proceedings of Machine Learning and Systems*, 2:14–29, 2020.

[15] Yonggan Fu, Haichuan Yang, Jiayi Yuan, Meng Li, Cheng Wan, Raghuraman Krishnamoorthi, Vikas Chandra, and Yingyan Lin. Depthshrinker: a new compression paradigm towards boosting real-hardware efficiency of compact neural networks. In *International Conference on Machine Learning*, pages 6849–6862. PMLR, 2022.

[16] Vinod Ganesan, Anwesh Bhattacharya, Pratyush Kumar, Divya Gupta, Rahul Sharma, and Nishanth Chandran. Efficient ml models for practical secure inference. *arXiv preprint arXiv:2209.00411*, 2022.

[17] Karthik Garimella, Zahra Ghodsi, Nandan Kumar Jha, Siddharth Garg, and Brandon Reagen. Characterizing and optimizing end-to-end systems for private inference. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ASPLOS 2023, page 89–104, New York, NY, USA, 2023. Association for Computing Machinery.

[18] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. *Advances in Neural Information Processing Systems*, 33:16961–16971, 2020.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[21] Chengcheng Huang, Xiaoxiao Dong, Zhao Li, Tengteng Song, Zhenguo Liu, and Lele Dong. Efficient stride 2 winograd convolution method using unified transformation matrices on fpga. In *2021 International Conference on Field-Programmable Technology (ICFPT)*, pages 1–9. IEEE, 2021.

[22] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, 2022.

[23] Siam Umar Hussain, Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar. Coinn: Crypto/ml codesign for oblivious inference via neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3266–3281, 2021.

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

[25] Nandan Kumar Jha and Brandon Reagen. Deepreshape: Redesigning neural networks for efficient private inference. *arXiv preprint arXiv:2304.10593*, 2023.

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

[27] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

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

[29] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *IEEE Symposium on Security and Privacy (SP)*, pages 336–353. IEEE, 2020.

[30] Souvik Kundu, Shunlin Lu, Yuke Zhang, Jacqueline Liu, and Peter A Beerel. Learning to linearize deep neural networks for secure and efficient private inference. *arXiv preprint arXiv:2301.09254*, 2023.

[31] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021, 2016.

[32] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305, 2019.

[33] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. Safenet: A secure, accurate and fast neural network inference. In *International Conference on Learning Representations*, 2021.

[34] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *ACM SIGSAC conference on computer and communications security*, pages 35–52, 2018.

[35] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[36] Hongwu Peng, Shanglin Zhou, Yukui Luo, Shijin Duan, Nuo Xu, Ran Ran, Shaoyi Huang, Chenghong Wang, Tong Geng, Ang Li, et al. Polympcnet: Towards relu-free neural architecture search in two-party computation based private inference. *arXiv preprint arXiv:2209.09424*, 2022.

[37] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. Secfloat: Accurate floating-point meets secure 2-party computation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 576–595. IEEE, 2022.

[38] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1003–1020. IEEE, 2021.

[39] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342, 2020.

[40] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 26–39. IEEE, 2021.

[41] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[43] Liyan Shen, Ye Dong, Binxing Fang, Jinqiao Shi, Xuebin Wang, Shengli Pan, and Ruisheng Shi. Abnn2: secure two-party arbitrary-bitwidth quantized neural network predictions. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 361–366, 2022.

[44] Wenting Zheng Srinivasan, PMRL Akshayaram, and Popa Raluca Ada. Delphi: A cryptographic inference service for neural networks. In *Proc. 29th USENIX Secur. Symp*, pages 2505–2522, 2019.

[45] Kevin Vincent, Kevin Stephano, Michael Frumkin, Boris Ginsburg, and Julien Demouth. On improving the numerical stability of winograd convolutions. 2017.

[46] Juan Yepez and Seok-Bum Ko. 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, 2020.