PolyKervNets: Activation-free Neural Networks For Efficient Private Inference

Toluwani Aremu, Karthik Nandakumar
Mohamed Bin Zayed University of Artificial Intelligence, UAE
toluwani.aremu@mbzuai.ac.ae, karthik.nandakumar@mbzuai.ac.ae

***Abstract*—With the advent of cloud computing, machine learning as a service (MLaaS) has become a growing phenomenon with the potential to address many real-world problems. In an untrusted cloud environment, privacy concerns of users is a major impediment to the adoption of MLaaS. To alleviate these privacy issues and preserve data confidentiality, several private inference (PI) protocols have been proposed in recent years based on cryptographic tools like Fully Homomorphic Encryption (FHE) and Secure Multiparty Computation (MPC). Deep neural networks (DNN) have been the architecture of choice in most MLaaS deployments. One of the core challenges in developing PI protocols for DNN inference is the substantial costs involved in implementing non-linear activation layers such as Rectified Linear Unit (ReLU). This has spawned a search for accurate, but efficient approximations of the ReLU function and neural architectures that operate on a stringent ReLU budget. While these methods improve efficiency and ensure data confidentiality, they often come at a significant cost to prediction accuracy. In this work, we propose a DNN architecture based on polynomial kervolution called *PolyKervNet* (PKN), which completely eliminates the need for non-linear activation and max pooling layers. PolyKervNets are both FHE and MPC-friendly - they enable FHE-based encrypted inference without any approximations and improve the latency on MPC-based PI protocols without any use of garbled circuits. We demonstrate that it is possible to redesign standard convolutional neural networks (CNN) architectures such as ResNet-18 and VGG-16 with polynomial kervolution and achieve up to $30\times$ improvement in latency of MPC-based PI with minimal loss in accuracy on many image classification tasks.**

***Index Terms*—Private inference, homomorphic encryption, multi-party computation, polynomial kervolution, non-linear activation, pooling**

I. Introduction

In today's cloud computing era, most major cloud service providers (CSP) offer machine learning (ML) as a service (MLaaS) to enable customers gain valuable insights from their data, without the need for investing in resources and expertise to build or run predictive models on their own. Deep neural networks (DNN) have become the de-facto standard for ML tasks, especially in the areas of computer vision and natural language understanding. Ever since the introduction of AlexNet [1] in 2012, which achieved state-of-the-art (SOTA) performance on a large-scale image classification task using a DNN model with 60 million parameters and tens of layers, the ML community has developed several deeper and larger DNN models. While SOTA vision models such as convolutional neural networks (CNN) and vision transformers have millions of parameters and hundreds of layers, SOTA language models such as GPT-3 have billions of parameters. These humongous models require extensive computational and memory resources, serving as a driving force for outsourcing/offloading ML computations to the cloud. Thus, the MLaaS model has the potential to solve many real-world problems in a cost-effective way across a range of industries including finance, healthcare, and logistics. For example, healthcare providers can exploit the MLaaS framework to accurately diagnose diseases based on patient data such as medical images and electronic health records.

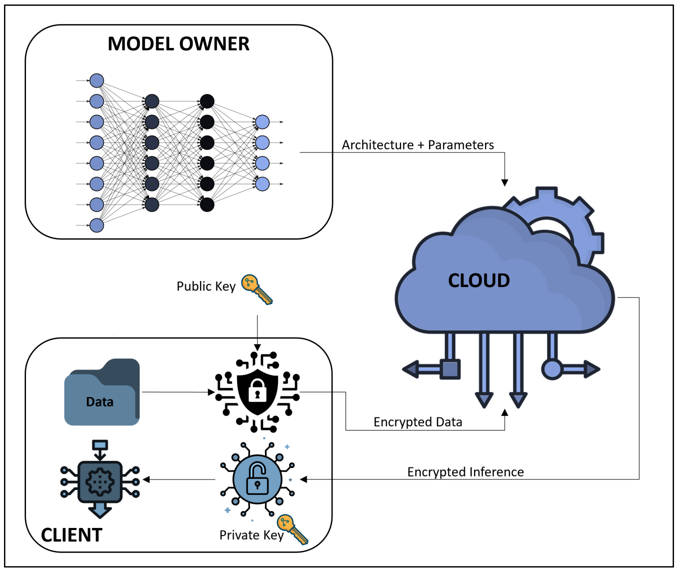

Fig. 1. An illustration of private machine learning inference based on fully homomorphic encryption.

The key roadblock in the widespread adoption of MLaaS is the sensitive nature of customer data and the privacy concerns associated with the release of such sensitive information to untrusted CSPs. Apart from end-user privacy concerns, regulations such as General Data Protection Regulation (GDPR) also place stringent data confidentiality requirements, thereby hindering the growth of MLaaS. Cryptographic tools such as homomorphic encryption (HE) and secure multi-party computation (MPC) offer a solution to the problem of privacy-preserving machine learning. While these confidential computing tools can be applied for both model learning and inference, they come at a tremendous computational and/or communication cost, often rendering the training task unviable. Therefore, most of the effort in privacy-preserving machine learning has focused on private inference (PI), where the goal is to apply a pre-trained ML model on "encrypted" data and perform inference in the encrypted domain (see Figure 1). In this context, encryption refers to the fact that the service provider never sees the private data of customers.

Fundamental challenges in the efficient implementation of PI protocols are the large depth of the underlying neural networks and the complexity introduced by the presence of non-linear (activation and pooling) layers in the DNN architecture. It is well-known that deeper networks have exponentially larger expressive power and better generalization ability compared to shallow architectures [32]. Furthermore, in the absence of non-linear activation layers, the entire DNN can be compressed into a single linear layer, severely degrading the accuracy of the resulting model. Thus, *both large depth and non-linearity are essential to the unparalleled success of DNN models* and compromising on them can cause substantial loss in accuracy of DNN models. However, *adding more layers and non-linearities stymie the possibility of designing efficient PI protocols*. This is because non-linear operations are orders of magnitude more computationally expensive in the case of both HE and MPC. Solving this accuracy vs. efficiency trade-off has been a subject of active research and the proposed solutions broadly fall into two categories. One approach is to develop accurate and efficient approximations for non-linear activation functions commonly used in DNNs such as Rectified Linear Unit (ReLU) [2]–[4]. Alternatively, neural architecture search (NAS) algorithms have been used to identify architectures that can work accurately within a constrained non-linearity (e.g. ReLU) budget [5]–[7]. While the first approach typically leads to significant accuracy degradation, the latter method caps the computational complexity, while maintaining accuracy at comparable levels.

In this work, we propose a different strategy to solve the accuracy vs. efficiency conundrum in PI protocols. Specifically, we leverage the idea of *kervolution* (kernel convolution) proposed in the computer vision literature to design DNNs that are more conducive to encrypted inference. Since kervolution operators are based on patch-wise kernel functions (as opposed to point-wise non-linearity in traditional CNNs), they can capture higher order interactions between features without increasing the number of parameters. Furthermore, kervolution also avoids the need for non-linear activation and max pooling layers - an ideal scenario for private inference. Therefore, the main contributions of this work are as follows:

- We are the first to identify the synergy between the requirements of private inference protocols and the characteristics of kervolutional neural networks [8], leading to the design of more accurate and efficient DNNs amenable to private inference.
- We modify the well-known CNN architectures (e.g., ResNet-18, VGG-16, etc.) to incorporate *polynomial kervolution* operators in lieu of traditional convolution, ReLU, and maxpooling layers. The resulting PolyKervNets (PKNs) lead to a minimal loss in accuracy compared to the baseline CNN architectures, when evaluated on four image classification datasets: CIFAR-10, CIFAR-100, MNIST, and Chest X-ray dataset for tuberculosis detection. However, we show that even this minimal loss in accuracy can be mitigated by designing shallower residual networks with polynomial kervolution.
- Finally, we leverage PolyKervNets to implement private inference based on secure multiparty computation (using the Cheetah protocol [33] and Delphi protocol [21]) on VGG-16 and ResNet-18 architectures. The results show that the proposed approach minimizes latency by up to $30\times$ on the Delphi protocol, with Cheetah giving similar results (but with much faster inference time). We also show preliminary results for FHE-based private inference (using the HELayers library [35]) based on the ResNet-10 architecture, which achieves better accuracy and lower latency compared to commonly used ReLU approximations.

II. Related Work

A. Private Inference

In a typical private inference pipeline, the client encrypts the raw data using his public key and sends the ciphertexts to the untrusted cloud. It is assumed that the model owner already has access to a pre-trained ML model. When the model owner also serves as the CSP, the pre-trained ML model is directly applied to the encrypted data and the encrypted inference results are sent back to the client for decryption using the secret key (as shown in Figure 1). In a more general case where the model owner is a separate entity (different from the CSP), the model architecture and encrypted model parameters (encrypted using client's public key) are sent to the cloud and the inference takes place in the encrypted domain.

There are two possible ways in which private inference can be achieved - fully homomorphic encryption (FHE) and secure MPC. When minimal communication between the client and CSP is desired, FHE is the only feasible solution. This approach has been enabled by the development of many FHE cryptosystems such as BFV [9], [10], BGV [11], CKKS [12], and TFHE [13] over the past decade. CryptoNets [2] was one of the earliest works to demonstrate that encrypted inference using FHE is possible. This was followed by other implementations such as Faster CryptoNets [14], Low Latency CrptoNets [15], Shift-accumulation based leveled HE (SHE) [16], and CKKS with bootstrapping [17]. Apart from SHE that uses gate-level HE operations based on TFHE, all the other implementations encounter difficulties in dealing with non-linear activation functions such as ReLU and max pooling layers. The typical work around is to replace ReLU with square activation [2] or other polynomial approximations [3], [4] and utilize sum pooling in the place of max pooling. As noted in [17], these choices lead to unacceptable trade-off between accuracy and efficiency.

An alternative approach is to rely on secure MPC or a combination of HE and MPC. Often, a two-party protocol (2PC) between the client and the CSP is used for this purpose. The key difference between the methods that fall under this umbrella is the manner in which the linear layers in DNNs are computed. While Gazelle [18] and SAFENet [19] use HE, MiniONN [20], DELPHI [21], CryptoNAS [5], and DeepReduce [7] rely on arithmetic secret sharing. As with FHE-based methods, the primary bottleneck in the MPC approach is the computation of non-linear layers, which is usually achieved using garbled circuits. Since garbled circuits are several orders of magnitude more expensive to compute than secret sharing protocols, non-linear computations take up the lion's share of the overall computational and/or communication load. This has led to the development of techniques that either perform ReLU dropping after training [7] or design neural network architectures that operate within a given ReLU budget. Hybrid methods that perform selective ReLU dropping as well as ReLU approximation have also been proposed [19].

In contrast to the above techniques, our goal is to completely remove all non-linear activation functions and almost all the pooling layers from the DNN architecture. This is facilitated by the use of kervolution operators, which was first proposed in [8]. To the best of our knowledge, our work is the first attempt to achieve efficient private inference through total elimination of activation and max pooling layers without compromising on accuracy.

B. *Kervolutional Neural Networks*

Existing CNN models primarily rely on the activation layers to introduce point-wise non-linearity into the ML model. In [8], it was argued that CNN architectures can be made more expressive if the linear convolution layer is replaced with patch-wise non-linearity using the kernel trick. This approach was referred to as kernel convolution (or kervolution) and has the potential to improve generalizability and model capacity, while retaining the advantages of simple linear convolution such as speed and number of parameters. Three kernel types were proposed in [8] for use in the kervolutional layer: Norm Kervolution, Polynomial Kervolution and Gaussian Kervolution. While both polynomial and Gaussian kernels were found to achieve good recognition accuracy in [8], Gaussian kervolution requires computing an exponential function, which is expensive in the encrypted domain. Hence, we utilize only polynomial kervolution (PolyKerv) in this work because of its better suitability to PI protocols.

Polynomial Kervolution: The standard convolution operator takes an input $\mathbf{x} \in \mathbb{R}^n$ (vectorized version of a 2-dimensional matrix) and filter weights $\mathbf{w} \in \mathbb{R}^n$ and outputs:

$$k_c(\mathbf{x}, \mathbf{w}) = (\mathbf{x}^T\mathbf{w} + b), \tag{1}$$

where b represents the bias. In contrast, the polynomial kervolution operator applies patch-wise non-linearity and outputs:

$$k_p(\mathbf{x}, \mathbf{w}) = (\mathbf{x}^T\mathbf{w} + c_p)^{d_p} = \sum_{j=0}^{d_p} c_p^{d_p - j}(\mathbf{x}^T\mathbf{w})^j, \tag{2}$$

where $d_p (d_p \in \mathbb{Z}^+)$ is the degree of polynomial and $c_p (c_p \in \mathbb{R}^+)$ is the balance factor. Intuitively, the polynomial degree extends the feature space to d_p dimensions and the balance factor is able to balance the non-linear terms. Thus, the polynomial kernel learns linear terms just like the standard convolutional operator, but is also able to output nonlinear terms, which can be considered as an approximation of an activation layer's output. In other words, polynomial kervolution subsumes the functions of both the standard convolution operator and the non-linear activation layer, while making the activation learnable and more expressive per use.

Though the idea of kervolution was introduced in [8], it was primarily used as a replacement for the standard convolution operator in CNNs, leaving the rest of the architecture (including activation and pooling layers) intact. The possibility of removing ReLU and replacing the max pooling layers with average pooling was left as a tantalising teaser in a single experiment, where the LeNet-5 architecture was modified to achieve good performance on the MNIST dataset. In this work, we push this idea to the extreme and attempt to design DNNs using only polynomial kervolution and completely removing all ReLU layers.

We believe that the use of kervolutional neural networks has not taken off in the broader computer vision community because of its poor cost-benefit trade-off. Replacing convolution with kervolution and retaining ReLUs increases the computational cost (for plaintext inference), while providing only marginal improvement in accuracy. Removing the ReLUs affects the stability of the training process and erases even the marginal accuracy gains. Hence, there is little incentive to use kervolutional neural networks for inference on plaintext data. However, the trade-offs are completely different in the private inference scenario. For private inference, removing ReLUs provides huge gains in latency and marginal reduction in accuracy becomes acceptable.

III. Proposed Method: PolyKervNets (PKNs)

The goal of this work is to design DNNs using polynomial kervolution layers that do not require non-linear activation functions and hence, more conducive for private inference. The challenge with designing such networks is the difficulty in training (optimizing their parameters) using traditional gradient descent techniques because of vanishing/exploding gradients. Since residual networks (ResNet) [22] are known to be easier to optimize, we start with ResNet as the baseline architecture and design **PolyKervResNets (PKRs)**, which are composed of kervolution-based residual blocks.

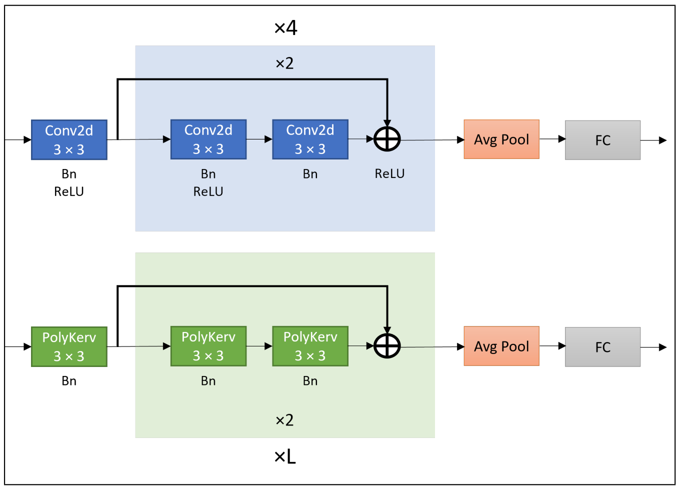

Fig. 2. Top: The residual block design of the standard (vanilla) ResNet-18 architecture. Bottom: The residual block design of PolyKervResNet (PKR-L). Here, L denotes the number of residual blocks, which is varied from 2 to 4, resulting in different PKR architectures with varying levels of depth.

A. *Input to the Residual Blocks*

A typical ResNet architecture consists of four residual blocks. However, before the input is fed to these residual blocks, it passes through a 7×7 convolution layer (with 64 channels and a stride of 2), followed by ReLU activation and a 3×3 max pooling layer (with stride of 2). In the PKR architecture, we modify the 7×7 convolutional layer into a 3×3 polynomial kervolution layer and remove the ReLU and max pooling layers which follows it (see Figure 2). This produces a 64-channel output that is a non-linear function of the input. We skip the ReLU and pooling layers because the kervolution layer already introduces the required non-linearity. Consequently, the size of the inputs to each residual block in a PKR is twice the size of the input to the corresponding residual block in a vanilla ResNet architecture.

B. *Residual Block Design*

Since our eventual target is to design DNNs that enable efficient private inference, we start with the ResNet-18 architecture. In ResNet-18, each residual block has two identical sub-blocks and each sub-block has the following structure: a 3×3 convolution layer, followed by ReLU activation, and another 3×3 convolution layer. The output of this second convolution layer is added to the output of an identity function (input to the block) and this combined output is passed through another ReLU activation. For the block, the first convolution has a stride of 2 which downsizes the input size by a factor of 2. The final output of the second sub-block serves as the input to the next residual block. In the proposed PolyKervResNet-18 (PKR-18), we eliminate all the ReLU activation layers within a residual sub-block and replace both the 3×3 convolution filters with 3×3 polynomial kervolution filters. This is illustrated in Fig 2, where L represents the number of residual blocks. There is no change in the number of filters/channels in any of the residual blocks. Note that it is also possible to design the residual sub-block in the PolyKervResNet such that only the first convolution layer is replaced with polynomial kervolution, the second 3×3 convolution layer remains unaltered, and the final ReLU function is eliminated. We refer to this variation as **MixedPolyKervResNet (MPKR)**.

C. *Output of the Residual Blocks*

Similar to vanilla ResNet, the output of the final residual block in a PKR is passed through an average pooling layer before it is flattened and presented to a fully connected (FC) layer. Since the PKR is trained on plaintext data, a softmax function is applied to the output of the FC layer and the network is trained using the standard cross-entropy loss. During private inference, the encrypted logit values from the final FC layer can be sent back to the client, who can decrypt them to perform inference (assign the sample to the class with the highest logit value).

D. *Other CNN Architectures*

To further establish the versatility and applicability of PolyKervNets, we replace all the convolution layers in other well-known CNN architectures such as VGG-16 [23], AlexNet [1] and LeNet-5 [24] with polynomial kervolution layers. We also remove all the non-linear activation layers in these networks and replace the max pooling layers with average pooling layers. The resulting architectures are referred to as **PolyKervVGG-16 (PKV-16), PolyKervAlexNet (PKA-8)** and **PolyKervLeNet-5 (PKL-5)**, respectively. Originally, all the above CNN architectures have three fully connected (FC) layers at the end, of which the first two FC layers are followed by activation layers and the last FC layer followed by a softmax layer. However, since our goal is to remove all the activation layers, we use only one FC layer followed by the softmax layer in our PolyKervNets (see Figure 3).

IV. Experimental Results

We demonstrate the effectiveness of PolyKervNets through experiments on four datasets (CIFAR-10, CIFAR-100, TB Chest X-ray, and MNIST) using four standard CNN architectures (ResNet-18, VGG-16, AlexNet, and LeNet-5). The proposed approach is benchmarked against five baseline methods: **ReLU** - this is original CNN architecture, **DeepReduce** - architecture obtained using the ReLU reduction method proposed in [7], **Square** - ReLU replaced by square activation [2], **PolyApprox-1** - ReLU replaced by quadratic polynomial approximation proposed in [3], and **PolyApprox-2** - ReLU replaced by quadratic polynomial approximation proposed in [4]. The evaluation metrics used for benchmarking are *accuracy* and *inference time/latency*. Note that the inference time in PI (especially FHE-based) protocols is highly dependent on

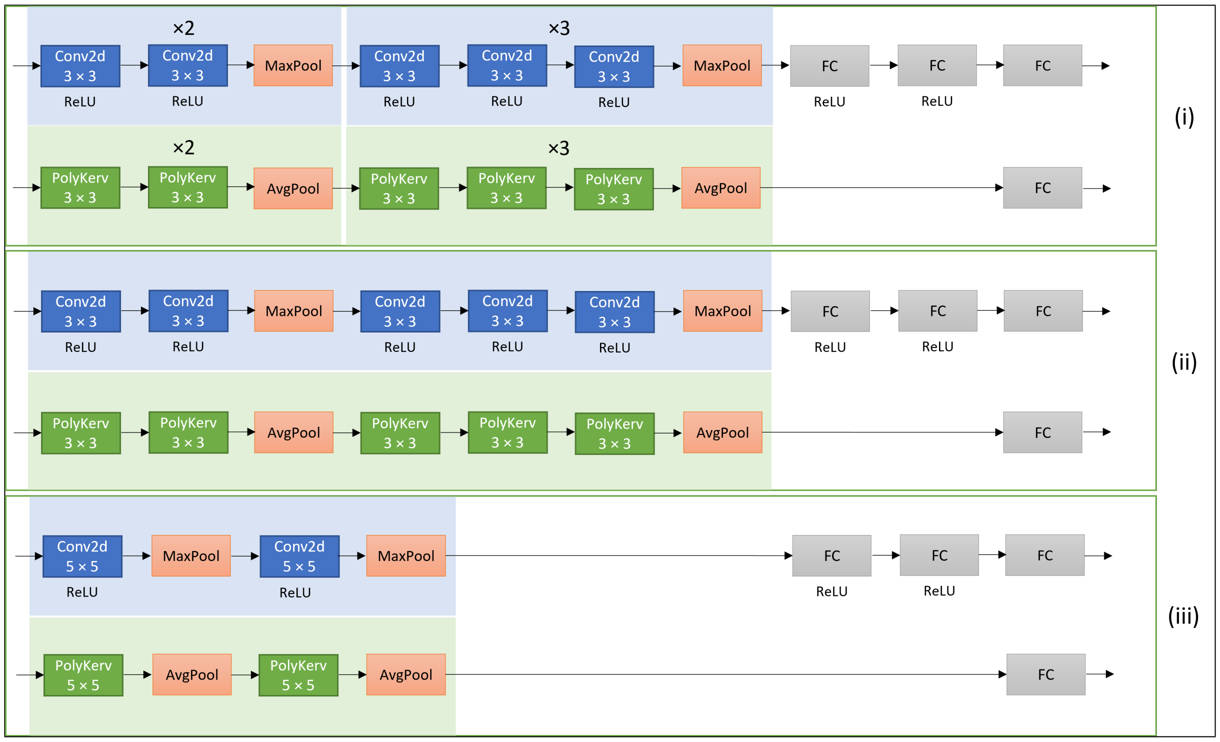

Fig. 3. Other CNN architectures that are redesigned as PolyKervNets. Box (i) shows the VGG-16 (blue) and PKV-16 (green) architectures; Box (ii) shows the AlexNet (blue) and PKA-8 (green) architectures; Box (iii) shows the LeNet-5 (blue) and PKL-5 (green) architectures.

the multiplicative depth of the ML model, which represents the maximum number of multiplicative operations (gates) required along any of its circuit paths. Hence, we also report the multiplicative depth of various models along with the number of parameters in the model, which is coarse proxy for model capacity.

A. Datasets

CIFAR10 and CIFAR100: CIFAR10 and CIFAR100 [26] are standard natural image classification datasets used in the literature. CIFAR10 contains 60000 images - 50000 images for training and 10000 images for inference. It consists of 10 classes and each class has an average of 5000 train images and 1000 test images. CIFAR100 also contains the same total number of images, with the only difference being that the 60000 images in CIFAR100 come from 100 classes at an average of 600 images per class. Both CIFAR10 and CIFAR100 contain RGB images of size 32×32.

TB Chest X-ray: In a bid to demonstrate the effectiveness of the proposed approach on a more real-world image classification task, we evaluate it on the tuberculosis (TB) Chest X-ray dataset [27]. This dataset has 4200 X-ray (grayscale) images of size 512×512 drawn from two classes. The normal class contains 3500 images, while the tuberculosis class contains 700 images.

MNIST: To demonstrate the working of a PolyKervNet for encrypted inference using the CKKS FHE scheme, we employed the much simpler MNIST dataset, which contains 60000 28×28 gray-scale images representing the 10 digits.

B. Pre-processing and Training Details

Our primary goal is to benchmark the accuracy and latency of PolyKervNets relative to the baseline CNN architectures. We would like to emphasize that the objective is not to achieve state-of-the-art accuracy on specific datasets. Therefore, we do not focus on searching for the best training settings for each architecture and dataset. Instead, we use common pre-processing steps, hyperparameter settings, and optimization algorithms across all architectures for a fair comparison.

CIFAR10 and CIFAR100: The CIFAR10 and CIFAR100 datasets have predefined train and test splits within the Pytorch dataset module. The input images were kept the same (32×32) and a batch size of 128 was used. We then employed horizontal flips and random rotation as data augmentation strategies for these two datasets. The whole train and test sets were also normalized before going in as inputs to the architectures.

TB Chest X-ray: Since there were no predefined train-test splits for this dataset, we implemented a custom dataset loader that randomly selects 3700 images for training and retains 500 images as the test set, while maintaining the original class proportions of the dataset. The images were downsized to 64×64 for faster training. The random crop augmentation was leveraged to extract a 56×56 patch from the downsized images and these cropped patches were resized back to a size of 64×64. A batch size of 32 was set for this task in order to have more training batches, while also reducing training time.

MNIST: No augmentation was applied for this dataset because the task is very simple. We chose a batch size of 32 for this dataset.

Training Details: All the DNN architectures were trained using the Adam optimizer with a learning rate of 1e-3 with a scheduler set to decrease the learning rate by 10 every 80 steps. For polynomial kervolution, we used polynomial degree $d_p = 2$ to ensure a fair comparison with square activation [2] and the quadratic polynomial ReLU approximations used in [4] and [3]. The balance factor c_p was set to 1 in our experiments. Each network is then trained for 200 epochs on CIFAR10 and CIFAR100, and 50 epochs on the TB Chest X-Ray datasets. All our experiments in the plaintext domain were run on the following computing environment:

- System used: NVIDIA QUADRO RTX 6000
- Operating System: Ubuntu 21.04
- GPU memory: 24 GB GDDR6

For private inference experiments based on for PolyKervNets (PKNs) as well as other baseline methods, we used the following computing environment:

- System used: Intel Xeon(R) Silver 4215 CPU
- Base Frequency: 2.50 GHz
- RAM: 32 GB
- PI protocols: HELayers CKKS [35], [36], Delphi [21], Cheetah [33]

TABLE I
TEST ACCURACY OF DIFFERENT RESNET-18 MODELS ON THE TB CHEST X-RAY, CIFAR-10, AND CIFAR-100 DATASETS.

Method	TB X-Ray (%)	CIFAR-10 (%)	CIFAR-100 (%)
ReLU [25]	99.2	91.9	72.8
DeepReduce [7]	99.2	92.4	72.5
PolyApprox-1 [3]	97.4	89.0	66.4
PolyApprox-2 [4]	98.5	89.4	68.3
Square [2]	97.1	84.2	61.7
PKR-18	99.2	90.1	71.3

C. Plaintext Evaluation

Accuracy of ResNet-18 models on different datasets: We first evaluate the accuracy of different ResNet-18 models, obtained using the five baseline methods and the proposed PolyKervResNet (PKR-18), on the TB Chest X-ray, CIFAR-10 and CIFAR-100 datasets. As emphasized earlier, we did not extensively search for the most optimal training settings for each dataset. Consequently, the accuracy achieved by the vanilla ResNet-18 model on CIFAR-10 (CIFAR-100) dataset was only 91.9% (72.8%). While these accuracy values are reasonably high, they do not match the state-of-the-art accuracy values reported in the literature. For instance, there are pre-trained ResNet-18 models available on the Internet that can achieve 93.07% accuracy on CIFAR-10[1] and 75.61% top-1 accuracy on CIFAR-100[2]. However, it is difficult in practice to replicate the exact training settings used to train such models. To enable fair comparison, it is essential to have a common training setting across different baseline methods and CNN architectures. Hence, we do not focus on optimizing the accuracy of each individual model and rather base our analysis on the relative accuracy trends.

From Table I, we observe that all models have high test accuracy on the TB X-ray dataset, with PKR-18 having the same accuracy of 99.2% as the original (ReLU) and ReLU-constrained (DeepReduce) models. On the CIFAR-10 and CIFAR-100 datasets, the ReLU and DeepReduce models had almost comparable accuracy, which was marginally higher than the PKR-18 model. However, the PKR-18 model had much better accuracy compared to square activation and marginally better accuracy compared to the polynomial ReLU approximations. This is because deep models with square activation often fail to converge well due to gradient explosion, indicating that square activation is not good enough to replace the ReLU function, especially in the case of complex classification tasks. This result is consistent with the observations reported in [4] on square activation. While the other polynomial approximations fare better, they still suffer from the same issues as square activation, albeit to a much lesser extent. Thus, rigid quadratic polynomial approximations are not effective replacements for the ReLU activation functions and the proposed PolyKervNets consistently outperform networks designed with such approximations.

The small accuracy degradation suffered by the PKR-18 model (in comparison to ReLU and DeepReduce) is primarily due to accumulation of errors during training, which is typically aggravated by the depth of the network. One way to mitigate this problem is to reduce the depth of the network, which can be easily achieved in a ResNet architecture by decreasing the number of residual blocks. The PKR-18 architecture uses the same number of residual blocks as ResNet-18 ($L = 4$). The model obtained by removing one residual block (PKR-14) has better accuracy compared to PKR-18 on CIFAR-10 as well as lower multiplicative depth and smaller number of parameters (see Table II). Further removing another residual block (PKR-10) again leads to performance improvement and resulting accuracy becomes almost equal to that of vanilla ResNet-18 with all the non-linear layers intact. Reducing the number of residual blocks to a value less than two did not yield any accuracy improvement, which can be explained by the excessive loss of model capacity. Thus, for the CIFAR-10 dataset, $L = 2$ represents the optimal number of residual blocks in a PolyKervResNet because it offers the best com-

[1]https://github.com/huyvnphan/PyTorch_CIFAR10
[2]https://github.com/weiaicunzai/pytorch-cifar100

promise between model capacity and ease of training.

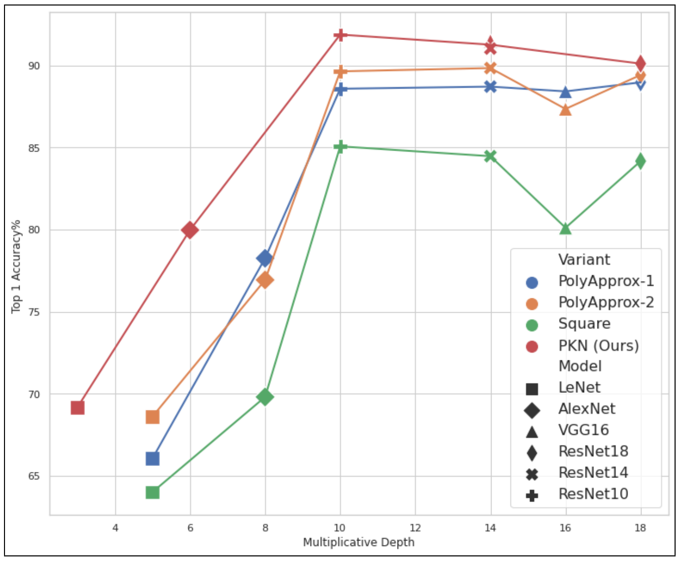

Fig. 4. Tradeoff between multiplicative depth and accuracy for various CNN architectures and baseline methods on the CIFAR-10 dataset. Clearly, polynomial kervolution achieves better trade-off compared to other ReLU approximations and ResNet-10 has the best accuracy among the different CNN architectures.

TABLE II
TEST ACCURACY ON THE CIFAR-10 DATASET, MULTIPLICATIVE DEPTH, AND NUMBER OF PARAMETERS FOR POLYKERVNETS.

Architecture	Accuracy (%)	Mult. Depth	Params (M)
PKR-18	90.1	18	11.2
PKR-14	91.0	14	2.8
PKR-10	91.9	10	0.7
PKV-16	91.2	14	14.8
PKA-8	79.9	6	3.4
PKL-5	69.2	3	0.007

Robustness to other CNN Architectures: The accuracy results for the various CNN architectures on the test split of the CIFAR-10 dataset are summarized in Figure 4 and Table II. There are three key pointers in these results:

- The PolyKerv versions of VGG-16, AlexNet, and LeNet-5 (referred to as PKV-16, PKA, and PKL-5, respectively) have slightly lower accuracy compared to their original (ReLU) counterparts but significantly higher accuracy compared to the rigid polynomial approximations of ReLU. At a high level, these results indicate the proposed idea of replacing convolution, ReLU, and max-pooling layers with polynomial kervolution and average pooling is generic and can be applied to most base CNN architectures, without much loss in accuracy.
- Figure 4 also shows that irrespective of the base architecture, polynomial kervolution achieves better trade-off between multiplicative depth and accuracy compared to polynomial approximations of ReLU. This again demonstrates the superiority of polynomial kervolution over ReLU approximations.
- Finally, among the various architectures, ResNet-10 (with two residual blocks) achieves the highest accuracy, even though it has a lower depth compared to ResNet-14, VGG-16, and ResNet-18. While the poor performance of shallow architectures like LeNet-5 and AlexNet are due to their small model capacity, the marginally lower accuracy of deeper architectures (ResNet-14, VGG-16, and ResNet-18) is due to the difficulty in training these networks in the absence of ReLU activations. For the selected dataset, ResNet-10 provides the best compromise between these two phenomena. Consequently, PKR-10 (which uses ResNet-10 architecture with polynomial kervolution) achieves the best accuracy on CIFAR-10, while having low multiplicative depth and fewer number of parameters.

TABLE III
TEST ACCURACY ON THE CIFAR-100 DATASET, MULTIPLICATIVE DEPTH, AND NUMBER OF PARAMETERS FOR VARIOUS MODELS.

Model	Method	Accuracy (%)	Mult. Depth	Params (M)
VGG-16	ReLU [25]	71.1	NA	34.0
	DeepReduce [7]	71.1	NA	34.0
	PolyApprox-1 [3]	67.6	16	33.6
	PolyApprox-2 [4]	68.1	16	33.6
	Square [2]	60.6	16	33.6
	PKV-16 (Ours)	70.2	14	14.8
ResNet-18	ReLU [25]	72.8	NA	11.2
	DeepReduce [7]	72.5	NA	11.2
	PolyApprox-1 [3]	66.4	18	11.2
	PolyApprox-2 [4]	68.3	18	11.2
	Square [2]	61.7	18	11.2
	PKR-18 (Ours)	71.3	18	11.2

The accuracy results for the various DNN architectures on the test split of the CIFAR-100 dataset are summarized in Table III. Due to the larger number of classes, we apply only the VGG-16 and ResNet-18 architectures on this dataset as the smaller architectures did not have sufficient model capacity to achieve good performance. Similar to the earlier results, the ReLU-constrained (DeepReduce [7]) ResNet-18 and VGG-16 models had almost the same test accuracy on CIFAR-100 as the corresponding vanilla models (which use ReLU activation without any constraints). However, these two methods are not amenable to FHE-based private inference because of the presence of ReLU functions. On the other hand, the FHE-friendly models with square activation [2] and polynomial approximations of ReLU [3], [4] have significantly less accuracy on CIFAR-100. In contrast, the proposed PolyKervNets (PKR-18 and PKV-16) are FHE-friendly and exhibit little compromise on accuracy, even though PKV-16 uses fewer number of parameters (due to the removal of two FC layers) compared to other VGG-16 models.

D. Private Inference Evaluation

MPC: We use the Cheetah [33] and Delphi [21] protocols to implement MPC-based private inference. Cheetah and Delphi are both hybrid cryptographic protocols that leverage a mix of 2PC and HE primitives for PI. Delphi uses HE operations

TABLE IV
PERFORMANCE OF VARIOUS MODELS FOR 2PC-BASED PRIVATE INFERENCE USING THE DELPHI AND CHEETAH MPC PROTOCOLS.

			DELPHI		CHEETAH	
Models-Dataset	Variants	Accuracy (%)	Latency (s)	Improvement	Latency (s)	Improvement
VGG-16 CIFAR-10	Vanilla	92.8	12.02		6.08	
	Deep Reduce	92.7	6.49	1.9×	3.93	1.5×
	PolyApprox-1	88.4	1.18	10.2×	0.42	14.7×
	PolyApprox-2	87.4	0.42	28.6×	0.36	16.9×
	Square	80.1	0.41	29.3×	0.36	16.9×
	PKV-16 (Ours)	91.2	0.38	31.6×	0.27	22.5×
	MPKV-16 (Ours)	85.3	0.30	40.1×	0.26	23.4×
VGG-16 TB Chest X-ray	Vanilla	97.8	39.59		10.34	
	PKV-16 (Ours)	96.2	1.10	36.0×	0.41	25.2×
	MPKV-16 (Ours)	94.8	1.07	37.0×	0.39	26.5×
ResNet-18 CIFAR-100	Vanilla	72.8	17.44		7.0	
	Deep Reduce	72.5	9.48	1.8×	4.76	1.5×
	PolyApprox-1	66.4	1.58	11.0×	0.49	14.3×
	PolyApprox-2	68.3	0.61	28.6×	0.43	16.3×
	Square	61.7	0.55	31.7×	0.42	16.7×
	PKR-18 (Ours)	71.3	0.59	29.6×	0.43	16.3×
	MPKR-18 (Ours)	70.5	0.53	32.9×	0.37	18.9×
ResNet-18 TB Chest X-ray	Vanilla	99.6	72.49		11.89	
	PKR-18 (Ours)	99.2	1.92	37.8×	0.79	15.1×
	MPKR-18 (Ours)	99.2	1.75	41.4×	0.69	17.2×

to implement the linear (convolution) layers, Beaver's multiplication triplets [31] for polynomial approximations and square activation, and Garbled Circuits (GC) [30] for non-linear computations such as ReLU. On the other hand, Cheetah is a state-of-the-art protocol that uses a novel HE-based (but SIMD-free) operation for the linear layers in DNNs and an improved Oblivious Transfer (OT) protocol for the nonlinear layers. The original Delphi work [21] also includes neural architecture search (NAS) to identify an optimal architecture, where some ReLU functions are replaced with polynomial approximations in order to reduce the amount of GC computations of ReLU. However, in this work the NAS is performed using the DeepReduce method [7], which has been proven to be more effective than the original Delphi NAS method.

We evaluated the inference time of our proposed method as well as other baselines on the TB-Chest Xray, CIFAR-10 and CIFAR-100 datasets using the VGG-16 and ResNet-18 architectures on each dataset. The accuracy and inference time of these models for 2PC-based private inference are summarized in Table IV. The vanilla version of VGG-16 (applied to CIFAR-10) and ResNet-18 (applied to CIFAR-100) implemented using the Cheetah protocol consumed about 6.1 and 7 seconds, respectively, for private inference. The DeepReduce method was able to achieve $1.5\times$ faster inference for both architectures without compromising on accuracy. On the TB-Chest Xray dataset, the vanilla version of both VGG-16 and ResNet-18 implemented using the Cheetah protocol consumed approximately 10.3 seconds and 11.9 seconds, respectively, for private inference.

Using the Delphi protocol, the vanilla version of both VGG-16 (applied to CIFAR-10) and ResNet-18 (applied to CIFAR-100) consumed about 12 seconds and 17 seconds, respectively, for private inference. The DeepReduce method based on the Delphi protocol achieved $1.9\times$ faster inference than the original VGG-16 and $1.8\times$ faster inference than the original ResNet-18 without compromising on accuracy. Clearly, the results for both protocols are consistent, with Cheetah being able to achieve faster PI than Delphi.

Among the approximation methods for both architectures, using both MPC protocols, the square activation has the best latency improvement (approximately $17\times$ over the original architectures while using Cheetah and $30\times$ while using Delphi). Unfortunately, the square activation also leads to a large degradation in accuracy. The PolyApprox-2 method of [4] achieves a much better accuracy-latency trade-off.

The proposed PolyKervNets clearly outperform all the above approximation methods and achieves both very good accuracy and large improvements in latency. For instance, complete private inference using the PKV-16 model can be obtained in 0.27 (0.38) seconds on the Cheetah (Delphi) protocol, making it about $22.5\times$ $(31.5\times)$ faster than the original VGG-16 architecture, while still achieving greater than 90% accuracy on CIFAR-10. Similarly, the PKR-18 model requires only 0.4 seconds and 0.6 seconds for Cheetah-based and Delphi-based PI on the CIFAR-100 dataset, while achieving an accuracy of approximately 71%. This is at least 3% absolute improvement in top-1 accuracy on CIFAR-100 over the comparable DeepReduce models having similar inference latency (despite the sub-optimal training settings used in this work, which results in lower accuracy of the original ResNet-18 model). We have also tested vanilla and PKN variants of ResNet-18 and VGG-16 on the TB-Chest X-ray dataset. Similar to the CIFAR datasets, the accuracy of PKN variants is marginally lower than the corresponding vanilla models, but with a significant reduction in latency. Thus, the proposed PolyKervNets have better accuracy-latency trade-off than state-of-the-art ReLU reduction/approximation methods.

For our Delphi-based PI experiments using DeepReDuce, we used a ReLU budget that gives the best performance while still having a fair improvement on latency. For ResNet-18 model on CIFAR-100 dataset, the DeepReduce model reported in our paper had a ReLU count of 248K, an accuracy of 72.53%, and a latency of around 9.5s. When the ReLU count is reduced to 98K, the accuracy degrades to 67.84% and the latency reduces to around 3.4s. In contrast, the PKR-18 model had an accuracy of 70.97% (based on the same dataset) and a latency of 0.6s. Thus, PKR-18 achieves almost 3% higher accuracy than DeepReduce, while being close to 6$\times$ faster.

Note that it is possible to further reduce the ReLU budget and obtain even faster inference using the DeepReduce method, but this comes at the cost of significant loss in accuracy. As reported in [7], among models that required less than a second for inference, the best achieved accuracy using the ResNet-18 architecture on CIFAR-100 dataset was around 68%. This is despite the fact that the original ResNet-18 model achieving an accuracy of 74.5% on CIFAR-100. Due to differences in the training settings, the corresponding accuracy of the original ResNet-18 model in our work on the same dataset is only 72.6%. Hence, we can expect ReLU-constrained ResNet-18 models with sub-second inference latency trained using our settings to have an accuracy significantly lower than 68% on CIFAR-100. Since a more detailed investigation of the accuracy vs. latency trade-off of DeepReduce has already been reported in [7], we have not repeated this experiment in this work.

The proposed PKR-18 achieved a latency of around 0.4 and 0.6 seconds per image for the CIFAR-100 dataset using the Cheetah and Delphi protocols, respectively. However, it must be noted that the maximum image size used in our evaluations was 64×64 (TB Chest X-ray dataset) and the corresponding latency based on both MPC protocols were 0.8 and 1.9 seconds. For real-world applications, higher resolution images (minimum of 224×224) need to be processed, which will increase the latency. But this latency increase can be mitigated through parallelization of computations across multiple GPUs/CPUs. Thus, the reduced latency of PKNs indeed brings them closer to practical feasibility.

FHE: We use the HELayers [35], [36] library for implementation of FHE-based private inference. HELayers optimizes the tensor packing strategy of various neural network computations to improve the latency of homomorphic encryption protocols deployed in the Microsoft SEAL [29] and HELib [37] libraries. The key advantage of this library is the ease of use through a Python API, while preserving efficiency by implementing most of its operations using C++. Unlike other popular python wrappers of SEAL such as TenSEAL [28], HELayers allows the use of multiple convolution layers (which is a consequence of the optimized ciphertext packing strategy enabled in this library). We implemented the various ResNet-10 variants used in this work on the TB Chest X-ray dataset and report the FHE-based inference results. Since the ReLU function cannot be directly implemented in FHE, the vanilla (ReLU) ResNet-10 model or the DeepReduce model cannot be used for FHE-based PI.

The ResNet-10 architecture is a shallower variant of the ResNet-18 architecture, but with only the first two residual blocks. The output of the second residual block is passed to the global average pool layer, which is then flattened and passed into a softmax layer that has two output neurons. The Adam optimizer with a learning rate of 1e-3 was used for training before encrypting the data to verify its private inference (PI) performance. The ResNet-10 architecture is employed instead of ResNet-18 to avoid the need for bootstrapping and to overcome memory limitations.

Table V shows the performance of the above architectures during encrypted inference. Though the competing architectures had comparable PI latency on the TB Chest X-ray dataset, the proposed PKR-10 and MPKR-10 architectures had the least degradation of accuracy. This experiment again confirms that polynomial approximations of ReLU are not good enough for deeper networks designed for complex real-world tasks and the real benefits (higher accuracy for approximately the same inference time) of PolyKervNets (PKNs) can be realized in such scenarios.

TABLE V
PERFORMANCE OF RESNET-10 VARIANTS FOR FHE-BASED PRIVATE INFERENCE USING HELAYERS.

ResNet-10	Accuracy (%)	Latency (s)
ReLU	96.8	NA
PolyApprox-1	95.1	307.89
PolyApprox-2	95.3	307.20
Square	94.7	307.19
PKR-10 (Ours)	96.1	307.20
MPKR-10 (Ours)	96.1	305.67

V. ABLATION STUDIES

A. Impact of c_p and d_p hyperparameters

While the core experiments focused on demonstrating the ability of PolyKervNets to preserve accuracy and improve latency, there are other factors which impact the performance of the proposed methods. Firstly, we attempted to study the impact of the balance factor c_p on the PolyKervNet (PKN) model performance. The accuracy of various PKN architectures on the CIFAR10 dataset using different values of c_p are shown in Figure 5. The balance factor c_p has a

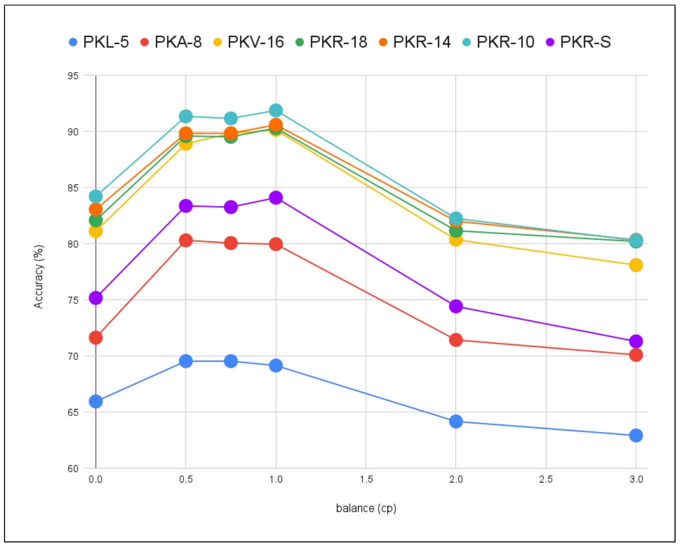

Fig. 5. Impact of the balance parameter (c_p) on the accuracy of each PolyKervNet (PKN) architectures, when evaluated on the CIFAR-10 dataset. We observe that the accuracy is highest when $c_p = 0.5$, $c_p = 0.75$ and $c_p = 1$.

significant impact on the accuracy of each model using a d_p of 2. However, $c_p = 0.5$, $c_p = 0.75$ and $c_p = 1$ gave the best accuracy and most stable performance for most architectures. Setting $c_p = 0$ could work well in smaller models and on simpler datasets, but does not work in the case of deeper models or on complex datasets. Using $c_p > 1$ could potentially enlarge each layer's output, and either explode the gradients or increase the loss, which results to a degradation in the mode performance. Using $0.5 \leq c_p \leq 1$ stabilizes the performance depending on the architecture and ensures a good accuracy when using a degree (d_p) of 2.

Next, we attempted to vary the polynomial degree d_p used in PolyKervNets. Using a d_p value of 3 or higher in PKN architectures exposes model training to the problem of exploding or vanishing gradients. Consequently, models with $d_p = 3$ failed to learn the tasks well and hence, the results are not reported in this work. Besides that, using a higher polynomial degree will increase multiplicative depth, thereby increasing latency. Hence, we believe that using a sufficient number of layers with a d_p value of 2 is good enough for most tasks.

B. PKNs vs. MPKNs

We performed a comprehensive study of MixedPolyKervNets and the results are summarized in Table VI. MixedPolyKervNets work well with the ResNet architecture, having marginally lower accuracy and lower latency compared to PolyKervNets. However, mixing convolutions and polynomial kervolutions does not work well with the VGG-16 architecture, indicating that residual connections play an important role in stabilizing the training of PolyKervNets.

TABLE VI
COMPARISON OF PKN AND MPKN ARCHITECTURES BASED ON CIFAR-10 DATASET.

PKN	Accuracy(%)	Latency(s)	MPKN	Accuracy(%)	Latency(s)
PKR-10	91.9	0.32	MPKR-10	91.9	0.25
PKR-14	91.0	0.45	MPKR-14	91.7	0.40
PKR-18	90.1	0.59	MPKR-18	89.0	0.53
PKV-16	91.2	0.38	MPKV-16	85.3	0.30

C. Impact of removing convolution layers

We also tried to reduce the depth of the network by reducing the number of convolution layers within each residual sub-block (from two to one). We refer to this architecture as PolyKervResNet-Small (PKR-S). Note that in our earlier experiments, the depth was reduced by decreasing the number of residual blocks instead of modifying the number of convolution layers within each block. Though the PKR-S architecture has the same multiplicative depth as PKR-10, the performance of the PKR-S model is still considerably lower than that of PKR-10. This justifies our original design of eliminating residual blocks as a whole to reduce depth, instead of reducing the number of convolutions within each sub-block.

TABLE VII
ABLATION SHOWING THE IMPACT OF KNOWLEDGE DISTILLATION ON THE ACCURACY OF PKR-18 ARCHITECTURES. HERE, WE OBSERVE AN IMPROVEMENT IN PERFORMANCE WHEN A PKN IS TRAINED BY A TEACHER MODEL USING DISTILLATION.

Teacher (%)	Model (cp, dp)	w/o Distillation (%)	w/ Distillation (%)
ResNet-18 (93.6%)	PKR-18 (cp=0, dp=2)	82.1	85.7
	PKR-18 (cp=0.5, dp=2)	90.0	92.4
	PKR-18 (cp=1, dp=2)	91.1	93.4
	PKR-18 (cp=2, dp=2)	81.1	85.1

D. Training using Knowledge Distillation

Instead of training from scratch, we tried to perform distillation using a pre-trained ResNet-18 model[3] as the teacher. Since this pre-trained model had a higher accuracy (93.6% on CIFAR-10) compared to our vanilla ResNet-18 model trained from scratch (91.89% on the same dataset), all the PKN variants distilled from this teacher had better accuracy compared to the corresponding models trained from scratch (see Table VII). For example, the distilled PKN-18 model had an accuracy of 93.41% compared to the same architecture trained from scratch (91.1%). This shows that distillation from a well-trained teacher model is an effective way of improving the accuracy of PolyKervNets. Unfortunately, we found that distillation had little impact on the sensitivity of the training process to other hyperparameters such as d_p.

VI. CONCLUSION

In this work, we attempted to design deep neural networks that are more suitable for private inference. Often, the presence of

[3]https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/resnet18d_ra2-48a79e06.pth

non-linear activation and pooling layers present the greatest stumbling block in achieving efficient private inference. In contrast to prevailing approaches that approximate non-linear layers such as ReLU with polynomial functions or remove as many non-linear computations as possible through neural architecture search, we propose a completely different approach based on polynomial kervolution. The proposed approach completely eliminates the need for activation layers and almost all pooling layers and leads to DNNs with low multiplicative depth, but good recognition accuracy.

While PolyKervNets are more amenable to private inference, it is important to note that training PolyKervNets is more unstable than traditional CNN models. This necessitates a careful selection of hyperparameters. On the issue of training stability, the key takeaways from this study are:

- While very shallow PolyKervNets are easy to train, the training stability decreases as the network depth increases. This is reason why PKR-10 provides better accuracy than PKR-14 and PKR-18. For the same reason, we have not been very successful (thus far) in training even deeper architectures such as ResNet-50.
- PolyKervNets with residual connections are relatively easier to train compared to other architectures such as VGG-16. Therefore, we recommend ResNet or models with residual connections as the core architecture for designing PolyKervNets.
- Across various architectures, $c_p = 1$ and $d_p = 2$ consistently give good performance.

Experiments on four datasets prove the efficacy of the proposed PolyKervNets based on both FHE and MPC techniques. Our experiments also show that reducing the multiplicative depth of neural networks is not a straightforward task of removing layers arbitrarily or using coarser approximations. Instead it is better to closely follow well-tested CNN architectures and modify them carefully to achieve better trade-off between accuracy and efficiency.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, Advances in Neural Information Processing Systems, 2012.

[2] 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 Proceedings of The 33rd International Conference on Machine Learning, 20–22 Jun 2016, pp. 201–210.

[3] N. Jain, K. Nandakumar, N. K. Ratha, S. Pankanti, and U. Kumar, “Efficient CNN Building Blocks for Encrypted Data”, CoRR, vol. abs/2102.00319, 2021.

[4] R. E. Ali, J. So, and A. S. Avestimehr, “On Polynomial Approximations for Privacy-Preserving and Verifiable ReLU Networks”, CoRR, vol. abs/2011.05530, 2020.

[5] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, “CryptoNAS: Private Inference on a ReLU Budget”, CoRR, vol. abs/2006.08733, 2020.

[6] Z. Ghodsi, T. Gu, and S. Garg, “SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud”, CoRR, vol. abs/1706.10268, 2017.

[7] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, “DeepReDuce: ReLU Reduction for Fast Private Inference”, CoRR, vol. abs/2103.01396, 2021.

[8] C. Wang, J. Yang, L. Xie, and J. Yuan, “Kervolutional Neural Networks”, CoRR, vol. abs/1904.03955, 2019.

[9] Z. Brakerski, “Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP”, in Advances in Cryptology – CRYPTO 2012, pp. 868–886.

[10] J. Fan and F. Vercauteren, “Somewhat Practical Fully Homomorphic Encryption”. 2012.

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(Leveled) fully homomorphic encryption without bootstrapping”, ACM Transactions on Computation Theory (TOCT), vol. 6, issue. 3, pp. 1–36, 2014.

[12] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers”, in International Conference on the Theory and Application of Cryptology and Information Security, 2017, pp. 409–437.

[13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “TFHE: Fast Fully Homomorphic Encryption Over the Torus”, Journal of Cryptology, 2019.

[14] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, “Faster CryptoNets: Leveraging Sparsity for Real-World Encrypted Inference”, CoRR, vol. abs/1811.09953, 2018.

[15] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, “Low latency privacy preserving inference”, in International Conference on Machine Learning, 2019, pp. 812–821.

[16] Q. Lou and L. Jiang, “SHE: A Fast and Accurate Privacy-Preserving Deep Neural Network Via Leveled TFHE and Logarithmic Data Representation”, CoRR, vol. abs/1906.00148, 2019.

[17] J.W. Lee, et al., “Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network”, IEEE Access, vol. 10, pp. 30039–30054, 2022.

[18] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, “GAZELLE: A Low Latency Framework for Secure Neural Network Inference”, in Proceedings of the 27th USENIX Conference on Security Symposium, Baltimore, MD, USA, 2018, pp. 1651–1668.

[19] Q. Lou, Y. Shen, H. Jin, and L. Jiang, “SAFENet: A Secure, Accurate and Fast Neural Network Inference”, in International Conference on Learning Representations, 2021.

[20] J. Liu, M. Juuti, Y. Lu, and N. Asokan, “Oblivious Neural Network Predictions via MiniONN Transformations”, in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, Texas, USA, 2017, pp. 619–631.

[21] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, “Delphi: A Cryptographic Inference System for Neural Networks”, in Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice, Virtual Event, USA, 2020, pp. 27–30.

[22] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition”, CoRR, vol. abs/1512.03385, 2015.

[23] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, in International Conference on Learning Representations, 2015.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition”, Proceedings of the IEEE, vol. 86, issue. 11, pp. 2278–2324, 1998.

[25] A.F. Agarap, “Deep Learning using Rectified Linear Units (ReLU)”, arXiv.org, 2018. https://arxiv.org/abs/1803.08375 (accessed Nov. 22, 2019).

[26] A. Krizhevsky, V. Nair, G. Hinton, “The CIFAR-10 and CIFAR-100 datasets (Canadian Institute for Advanced Research)”, http://www. cs. toronto. edu/kriz/cifar.html, 2009.

[27] T. Rahman, et al., “Reliable Tuberculosis Detection Using Chest X-Ray With Deep Learning, Segmentation and Visualization”, IEEE Access, vol. 8, pp. 191586–191601, 2020.

[28] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, “TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption”, arXiv [cs.CR]. 2021.

[29] Microsoft Research, “Microsoft SEAL (release 3.5)”, April, 2020. Available at: https://github.com/Microsoft/SEAL.

[30] A. C. Yao, “How to generate and exchange secrets”, 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), 1986.

[31] D. Beaver, “Precomputing oblivious transfer”, Advances in Cryptology — CRYPT0’ 95, pp. 97–109, 1995.

[32] R. Eldan and O. Shamir, O., "The power of depth for feedforward neural networks". Conference on learning theory, pp. 907-940, 2016.

[33] Z. Huang, W. Lu, C. Hong, and J. Ding, “Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference”, 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 809–826.
[34] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic Encryption for Arithmetic of Approximate Numbers”, Advances in Cryptology – ASIACRYPT 2017, 2017, pp. 409–437.
[35] E. Aharoni et al., “HELayers: A tile tensors framework for large neural networks on encrypted data,” arXiv [cs.CR], 2020.
[36] E. Aharoni et al., ”Complex Encoded Tile Tensors: Accelerating Encrypted Analytics,” IEEE Security & Privacy, vol. 20, no. 5, pp. 35-43, 2022.
[37] S. Halevi and V. Shoup, “Design and implementation of HElib: a homomorphic encryption library,” Cryptology ePrint Archive, 2020.