Fast and Private Inference of Deep Neural Networks by Co-designing Activation Functions

Abdulrahman Diaa*[1], Lucas Fenaux*[1], Thomas Humphries*[1], Marian Dietz[1], Faezeh Ebrahimianghazani[1], Bailey Kacsmar[1], Xinda Li[1], Nils Lukas[1], Rasoul Akhavan Mahdavi[1], Simon Oya[1], Ehsan Amjadian[1, 2], and Florian Kerschbaum[1]

[1]University of Waterloo
[2]Royal Bank of Canada
{abdulrahman.diaa, lucas.fenaux, thomas.humphries, marian.dietz, f5ebrahi, bkacsmar, xinda.li, nlukas, rasoul.akhavan.mahdavi, simon.oya, ehsan.amjadian, florian.kerschbaum}@uwaterloo.ca

Abstract

Machine Learning as a Service (MLaaS) is an increasingly popular design where a company with abundant computing resources trains a deep neural network and offers query access for tasks like image classification. The challenge with this design is that MLaaS requires the client to reveal their potentially sensitive queries to the company hosting the model. Multi-party computation (MPC) protects the client's data by allowing encrypted inferences. However, current approaches suffer from prohibitively large inference times. The inference time bottleneck in MPC is the evaluation of non-linear layers such as ReLU activation functions. Motivated by the success of previous work co-designing machine learning and MPC, we develop an activation function co-design. We replace all ReLUs with a polynomial approximation and evaluate them with single-round MPC protocols, which give state-of-the-art inference times in wide-area networks. Furthermore, to address the accuracy issues previously encountered with polynomial activations, we propose a novel training algorithm that gives accuracy competitive with plaintext models. Our evaluation shows between 3 and 110× speedups in inference time on large models with up to 23 million parameters while maintaining competitive inference accuracy.

1 Introduction

The rapid development of increasingly capable machine learning (ML) models has resulted in significant demand for products like machine learning as a service (MLaaS). In this scenario, big tech companies with vast computing resources train large machine learning models and provide users with query access. The major pitfall with MLaaS is that it requires clients to submit potentially sensitive queries to an untrusted entity. A promising solution to this problem is to employ cryptography to ensure the queries and inferences are hidden from the model owner. Secure inference is an active field of research with many solutions and different threat models as summarized in a recent SoK [32]. The challenge is that despite recent advances, the inference times are still prohibitively large compared to plaintext inferences.

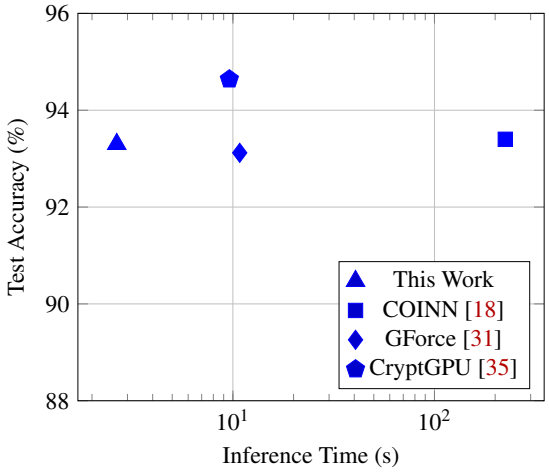


Figure 1: Summary of the inference time in seconds vs. test accuracy for each state-of-the-art approach on the CIFAR-10 dataset in the WAN (100 ms roundtrip delay).

This work focuses on reducing the runtime of secure inference on image data, under realistic network conditions, while maintaining classification accuracy. We consider the two-party setting using multi-party computation (MPC), where the server holds the modified ML model, and the client holds the data to query the model. Recent state-of-the-art works in this space employ various co-design approaches to reduce the inference time [32]. For example, COINN co-designs ML models optimized for quantization with efficient MPC protocols tailored to the custom models [18]. COINN substantially compresses the model and makes numerous optimizations to the architecture to achieve fast inferences. Another example is GForce, which tailors the cryptography needed for ML to high-speed GPU hardware [31]. By offloading vast

*Equal contribution

amounts of work to the pre-computation phase, they are able to achieve state-of-the-art runtime and accuracy in secure inference [31]. Similarly, CryptGPU [35] modifies the CrypTen framework [22] to run efficiently on the GPU and give state-of-the-art inference times in wide area networks. However, despite making major steps towards practical inference, none of these works remove a crucial bottleneck in secure inference: the non-linear layers.

It is well known that the non-linear layers are the bottleneck of secure inference [12, 13, 18, 29]. This is because secure computation on arithmetic shares is optimized for multiplications and additions, instead of non-linear layers such as ReLU activation functions or MaxPool layers. In order to compute these non-linear functions, expensive conversions between different types of MPC protocols are required. Specifically, in more realistic network settings with high latency, the inference time is substantially degraded due to each conversion taking many rounds of communication. This problem is particularly prevalent in deep neural networks (DNNs), where a non-linear activation separates each of the many linear layers.

This work addresses the non-linear layers by taking a co-design approach between the activation functions and MPC. We take the approach of replacing classic ReLU activation functions with a polynomial approximation during training and inference to avoid conversions altogether. Previous work has considered this approach but with limited success [12]. We propose two modifications to make this approach practical. First, we develop and evaluate new single-round MPC protocols that give the fastest evaluation of polynomials to date. The challenge with using polynomials is they severely impact model accuracy [12, 18, 29]. Previous work could not successfully train DNNs with more than 11 layers due to exploding gradients [12]. Thus, our second contribution is tailoring the ML training process to ensure high accuracy and stable training using polynomials. Our approach utilizes a new type of regularization that focuses on keeping the input to each activation function within a small range. We achieve close to plaintext accuracy on models as deep as ResNet-110 [16] and as large as a ResNet-50 on ImageNet [10] (23 million parameters). The combination of these approaches yields a co-design with state-of-the-art inference times and the highest accuracy for polynomial models.

We compare our work with three solutions representing the state-of-the-art approaches in secure inference according to Ng and Chow [32]. We summarize our results in Figure 1. Combining the single-round MPC protocols with our activation regularization achieves significantly faster inference times than all other solutions. Specifically, our solution is faster than CryptGPU by 4×, GForce by 5×, and COINN by 40× on average in wide area networks. Our approach also scales to large models on ImageNet with a 110× speedup over COINN, 14× speedup over Cheetah, and a 3× speedup over CryptGPU. Furthermore, our inference accuracy remains competitive with all other solutions. CryptGPU often gives slightly higher accuracy as it can evaluate any plaintext model (albeit slower than our work). Thus, the challenge for future work is to further close the ML accuracy gap between plain and polynomial models.

2 Background

2.1 Multi Party Computation

Secure multi-party computation (MPC) allows a set of parties to jointly compute a function while keeping their inputs to the function private. We focus on a variant of MPC which performs operations over shares of the data [5]. We use $[[s]] = [[s]_A, [s]_B]$ to denote that the value of s is shared among participants, where $[s]_A$ is the share held by party A and $[s]_B$ by party B. Arithmetic MPC protocols utilize a linear secret sharing scheme, such as an additive secret sharing scheme to compute complex circuits using combinations of additions and multiplications. Given constants v_1, v_2, v_3 and shares of values $[[x]], [[y]]$, one can locally compute

$$v_1[[x]] + v_2[[y]] + v_3 = [[v_1 \cdot x + v_2 \cdot y + v_3]] \tag{1}$$

to obtain shares of the value $v_1 \cdot x + v_2 \cdot y + v_3$. For multiplication, one can use Beaver's trick to multiply using a single round of communication between parties [4]. Specifically, we assume a triplet of random numbers a, b, c (called a Beaver triplet) was generated such that $a \cdot b = c$ and secret shared among all parties ahead of time (typically in an offline pre-computation phase). Then the parties compute $[[x \cdot y]]$ by first locally computing $[[a + x]] = A$ and $[[b + y]] = B$ and reconstructing A and B so that both parties have them in plaintext. This reconstruction is the single round required. Using these values, the parties compute the result locally as $[[x \cdot y]] = A[[y]] + (-B)[[a]] + [[c]]$, using the linearity property in equation 1.

Arithmetic MPC protocols are limited to basic multiplications and additions. Thus, for computing non-linear operations such as comparisons, other techniques such as converting to binary secret shares or using Yao's garbled circuits are common [9]. A binary secret sharing scheme is an arithmetic scheme carried out bitwise in the ring $\mathbb{Z}_2$. Specifically, the difference is that we first decompose x into its bits and have a separate arithmetic share of each bit. By maintaining this bitwise structure, operations such as XOR or bit shifts are trivial. We describe how to use a binary and arithmetic secret-sharing scheme together to compute non-linear functions in Section 3.3.

2.2 Neural Network Inference

We consider DNN classifiers with domain $\mathcal{X} \subseteq \mathbb{R}^d$ and range $\mathcal{Y} \subseteq \mathbb{R}^c$. DNN classifiers consist of a sequence of layers, each performing either a linear or a non-linear operation. The

ResNet [16] architecture we consider is composed of (i) convolutional, (ii) fully connected, (iii) pooling, (iv) batch normalization, and (v) ReLU layers. All layers are linear, except for $\text{ReLU}(x) = \max(x,0)$ and max pooling (that can be replaced with average pooling). To classify an input x, a classifier h, passes the input sequentially through each layer. Upon reaching the last layer, the prediction is obtained by taking $\arg\max_{i\in\{1..c\}} h(x)_i$, where we call h the logit function for a classifier $h : \mathcal{X} \to \mathcal{Y}$. Our work is tailored to securely perform the inference phase. Specifically, for an encrypted input x we compute the encrypted output $h(x)$ of the logit function. However, to achieve improvements during inference, our work requires modification to the training phase that generates h.

3 Problem Setup and Motivation

3.1 Problem Setup

We follow the same threat model as prior work for two-party secure inference [18, 22, 29, 31]. Specifically, we follow the two-party client-server model where the server has a machine learning model (a DNN) they have trained (with our technique), and the client holds private data upon which they would like to make an inference. Like Delphi [29], our work assumes the server uses a modified training procedure. The server's input to the protocol is the weights of their trained model, which they do not want to leak to the client (due to intellectual property or protecting their MLaaS business [36]). The client has a private input (typically an image) they would like to classify using the model but do not want to leak this input or the prediction to the server. That is, the MPC function can be written as $f(\text{image}, \text{model}) = (label, \emptyset)$. Following previous work [18, 22, 29, 31], we consider the semi-honest model [15, §7.2.2], where adversaries do not deviate from the protocol but may gather information to infer private information. Also, in line with previous work, we assume the model architecture is known to both parties. This includes the dimensions and type of each layer, parameters such as field size used for inference, and the mean and standard deviation of the training set [22].

Our goal is to reduce the inference time as much as possible. We are willing to incur a small degradation in accuracy to achieve practical runtimes. For example, in the streaming setting, applications like spam detection are runtime-critical (a small accuracy trade-off can be tolerated to make it feasible). Since all the protocols we compare with contain pre-computation, we focus on the online phase for fair comparison. Furthermore, the online time determines the latency, which we focus on reducing in this work. We build from CrypTen [22], which does not implement pre-computation and rather assumes a third-party dealer for Beaver triplets. However, in practice, the server and client could generate the Beaver triplets in a pre-computation phase using off-the-shelf protocols [21, 33].

3.2 Privacy During Model Training

We focus only on the inference phase of machine learning. However, the privacy of the training process and training data is an orthogonal but essential problem. We recommend that the data owner take appropriate steps to protect the privacy of the model, such as training using differential privacy [2] or rounding the output of the inference. Furthermore, during training, care should be taken to protect against threats such as model stealing, which can be launched using only the inference result [19]. To summarize, the model owner learns nothing other than the fact a query was made. We ensure only the inference is revealed to the client; however, ML attacks that only require black box query access [19] must be defended against during the training process. Since we focus the effects of MPC on the runtime and accuracy of ML inference, we did not conflate this comparison with additional privacy preserving training goals. Any privacy-preserving technique would add a similar overhead (e.g., reducing the accuracy) to all approaches we evaluate.

3.3 Motivating the Co-Design of Activation Functions

It has been well established in the literature that activation functions such as ReLU are the bottleneck in MPC-based secure inference, taking up to 93% of the inference time [12, 13, 18, 29]. The reason for this is that current approaches use different types of MPC protocols for a model's linear and non-linear layers [18, 20, 22, 29, 31]. The linear layers are typically computed using standard arithmetic secret-sharing protocols tailored for additions and multiplications. The non-linear layers are computed using garbled circuits or binary secret share-based protocols. The bottleneck in wide area networks is typically the conversions between these protocols as they require a large number of communication rounds. A typical DNN architecture has many linear layers, each followed by a non-linear layer resulting in a prohibitively large number of conversions.

Consider CrypTen, a PyTorch-based secure ML library, as a baseline approach [22]. CrypTen uses binary shares to evaluate boolean non-linear layers such as ReLUs and MaxPooling layers. Specifically, all linear layers are computed using standard multiplication and addition protocols over arithmetic shares. To compute $[[ReLU(x)]]$ at each layer, $[[x]]$ is first converted to binary shares using a carry look-ahead adder. Once in binary shares, CrypTen extracts the sign bit to compute $[[x > 0]]$ (a local operation). The sign bit, $[[x > 0]]$, is then converted back to arithmetic shares (trivial for a single bit) and multiplied with $[[x]]$ to get $[[ReLU(x)]]$. The problem with this approach is that each conversion takes $O(log(L))$ communication rounds. Taking into account the additional round needed for multiplication, we observe nine communication rounds per ReLU in practice (under 64-bit precision). Re-

cently, more sophisticated MPC protocols have been proposed that reduce the number of rounds needed for comparisons in arithmetic shares [6, 7] or reduce the cost of binary share conversions [11]. However, even if one were to implement these protocols in CrypTen, the number of rounds needed for non-linear layers would still outweigh the number needed for linear layers.

Motivated by this bottleneck, several works have focused on either reducing the number of ReLUs or replacing ReLUs altogether [12, 13, 20, 24, 29, 30]. One approach is to approximate each ReLU with a high degree polynomial [12, 24]. The advantage of using polynomials is that polynomials can be computed using arithmetic shares, thus removing the need for expensive conversions and improving the total inference time. A significant challenge with polynomials is maintaining model accuracy [12, 13, 18, 20, 29].

Thus, this work aims to provide a secure inference protocol with state-of-the-art inference time and accuracy in realistic networks with high latency. To do this, we take a co-design approach to balance accuracy and fast inference time. In Section 4, we develop MPC protocols that achieve the fastest evaluation of polynomials to date, assuming a modified ML architecture. In Section 5, we tailor the ML training procedure to achieve high accuracy using this modified architecture.

4 Faster Evaluation of Polynomials

In this section, we evaluate the speed-up of replacing ReLU's with a naive polynomial approximation. We then develop our single-round protocols and show that they drastically reduce the activation function evaluation time in wide area networks.

4.1 The Polynomial Advantage

To highlight the speed-up of polynomials over standard ReLUs, we first evaluate the runtime of a single layer with 2^{15} ReLU activation functions in Figure 2. (See Section 6 for more implementation details.) First, we plot an unmodified version of CrypTen (using CryptGPU [35]) with the conversion to binary shares. Next, we replace the ReLU with a degree four polynomial fitted using least squares polynomial regression (see Section 5 for the details). We can see that using polynomials in off-the-shelf CrypTen is much faster across all network speeds than the default mixed arithmetic and binary protocol. This difference becomes more pronounced as we add more network delay or scale to deeper models with more ReLUs.

Despite the significant speed-up, naively computing a polynomial is still expensive in MPC with a non-trivial number of communication rounds. For example, Horner's method (an iterative approach to evaluating polynomials) uses $O(n)$ communication rounds (where n is the degree of the polynomial). However, most MPC libraries (including CrypTen) use the square-and-multiply algorithm for exponentiation, followed

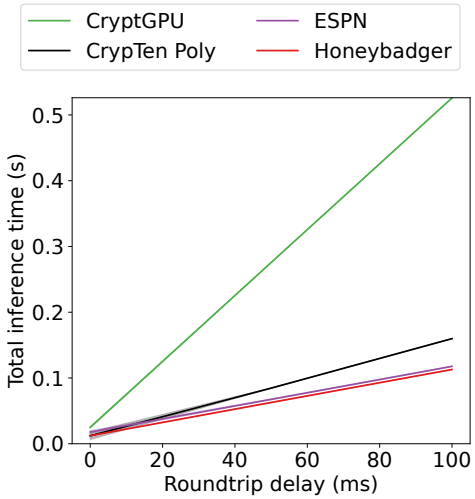


Figure 2: Benchmarking the secure evaluation of ReLU activation functions using various approaches. The x-axis is the network delay in ms and the y-axis is the mean runtime in seconds averaged over 20 runs with the shaded area representing the 95% confidence intervals.

by multiplying and summing the coefficients locally. The square-and-multiply algorithm requires $O(log(n))$ multiplications (and thus rounds) in MPC. In practice, the default square and multiply implementation in CrypTen uses two rounds per ReLU for a degree four polynomial. To increase the advantage of using polynomial activation functions even further, we develop a new single-round protocol for evaluating polynomials in MPC.

4.2 ESPN: Exponentiating Secret Shared Values using Pascal's triangle

We present our single-round, highly parallelizable protocol ESPN for computing high-degree polynomials. The fundamental idea is utilizing the binomial theorem (Pascal's triangle) to achieve faster exponentiation. We begin by describing our protocol for raising a number $[[x]]$ to the power k, in MPC (see Algorithm 1 for an overview). Using the additive secret sharing scheme, the exponentiation corresponds to $(x_A + x_B)^k$ where x_A represents the first party's share and x_B represents the second (such that $x_A + x_B = x$). The binomial theorem expands this expression as:

$$x^k = (x_A + x_B)^k = \sum_{i=0}^{k} \binom{k}{i} x_A^{k-i} x_B^i \tag{2}$$

We observe that, for each i in the sum, party A can compute $\mathbf{a}_i = x_A^{k-i}$ without needing to communicate with party B (Alg. 1 line 4). Similarly, party B can compute x_B^i without communicating with party A (Alg. 1 line 5). Finally, $\binom{k}{i}$ can be computed by any party (or pre-computed ahead of time).

For simplicity, we assign the computation of $\binom{k}{i}$ to party B. Thus party B computes $\mathbf{b}_i = \binom{k}{i} x_B^i$.

Once each party has computed their respective vectors, we multiply $\mathbf{a}_i \cdot \mathbf{b}_i$ for each i in parallel (Alg. 1 line 6). We carry out this multiplication using standard MPC protocols in one round. To use these multiplication protocols, each party must have a share of the input. We use a trivial additive secret sharing, where the other party inputs zero as their share to the protocol (Alg. 1 line 2). Finally, after the multiplication, the sum of the binomial theorem can be efficiently computed with no communication (Alg. 1 line 7).

Algorithm 1 Exponentiation Protocol for 2-party additive secret-sharing

1: **procedure** $Exp([[x]], k)$
2: $\quad \mathbf{a} = [[\mathbf{0}^k]]$, $\mathbf{b} = [[\mathbf{0}^k]]$ ▷ Initialize shares
3: $\quad$ **for** $i = 0 : k$ **do**
4: $\quad\quad$ Party A computes: $\mathbf{a}_i = \mathbf{a}_i + x_A^{k-i}$
5: $\quad\quad$ Party B computes: $\mathbf{b}_i = \mathbf{b}_i + \binom{k}{i} x_B^i$
6: $\quad \mathbf{p} = BeaverMultiply(\mathbf{a}, \mathbf{b})$ ▷ Parallel Multiplication
7: $\quad s = \sum_{i=0}^{k} \mathbf{p_i}$ ▷ $s = \mathbf{a} \cdot \mathbf{b}$
8: $\quad$ **return** s ▷ secret-shares of x^k

4.3 Polynomial Evaluation with ESPN

Floating Point Considerations. Our initial description of ESPNconsiders the integer domain for simplicity. Extending to floating point values is straightforward but requires rescaling (a standard practice in fixed-point arithmetic). We use CrypTen's two-party, local truncation protocol to ensure we do not incur additional rounds. However, there is a negligible chance of an incorrect result from this truncation protocol due to wrap-around in the ring. Specifically, the probability of an incorrect result when truncating x is $\frac{|x|}{2^L}$ where 2^L is the size of the ring [22, Appendix C.1.1]. This implies that x must be small compared to the ring for this fast truncation protocol to be correct.

Polynomial Evaluation Protocol. Using our exponentiation protocol, we show how to compute high-degree polynomials efficiently in a single round. We overview the protocol in Algorithm 2. First, in parallel, we compute all needed exponents using Algorithm 1. We recall that to ensure the correctness of truncation, we must ensure all intermediate values remain small. For simplicity, we define small to be that no intermediate scale becomes larger than twice the working precision p. We show the complete failure probability calculations of Algorithm 2 in Appendix E.

To ensure the values remain small after exponentiation, we create multiple scaled-down copies of the input x, proportional to each exponent we need to calculate. To compute x^i we first scale x down by $2^{-\bar{s}}$ where $\bar{s} = \lceil (i-2)p/i \rceil$) and p is the current working precision of x (line 5). $\bar{s}$ is chosen such that 2^p becomes approximately $2^{2p/i}$ after scaling and thus approximately 2^{2p} after exponentiation by i. These are approximations as not all values of i divide $(i-2)p$, so there is some error from taking the ceiling. To account for the additional factor of approximately two, we do an additional rescaling after each exponentiation in line 7. This rescaling incorporates $\bar{s}$ (which includes the ceiling function) to ensure all values are scaled back to 2^p. Finally, after computing all powers, we can locally multiply the result by the coefficients (public values) and sum in line 8. We give a complete proof of correctness for Algorithm 1 and 2, including the truncation operator in Appendix E.

Algorithm 2 Polynomial Evaluation Protocol for 2-party additive secret-sharing

1: **procedure** $Poly([[x]], \alpha, n)$
2: $\quad \mathbf{p}_1 = x$
3: $\quad$ **for** $i = 2 : n$ **do** ▷ In parallel
4: $\quad\quad$ Let $\bar{s} = \lceil (i-2)p/i \rceil$ ▷ Scale down factor
5: $\quad\quad x_i' = x \cdot 2^{-\bar{s}}$ ▷ Scale down before Exp
6: $\quad\quad \mathbf{p}_i = Exp(x_i', i)$ ▷ From Algorithm 1
7: $\quad\quad \mathbf{p}_i' = \mathbf{p}_i * 2^{-p*(i-1)+\bar{s}\cdot i}$ ▷ Scale down after Exp
8: $\quad y = \alpha_0 + \sum_{i=1}^{n} \alpha_i \cdot p_i'$ ▷ Locally dot product
9: $\quad$ **return** y ▷ Secret-shares of $f(x)$

Hyperparameter Restrictions. While Algorithm 2 is designed to keep intermediate values small, multiple hyperparameters determine the protocol's effectiveness. Using results from Appendix E (namely the max of Theorem E.2 and Theorem E.3), we get that the probability of failure for a given truncation is bounded by

$$Pr[\text{Truncation Failure}] \leq \frac{2^{n(\lceil \log_2 \lambda \rceil + 1) + 2p}}{2^L}. \tag{3}$$

For our experiments, we use CrypTen, with $L = 64$ [22] and a default working precision of $p = 16$. Assuming default values of $n = 4$ and $\lambda = 5$ we get a failure probability bound of 2^{-16}. However, this is a pessimistic upper bound since we consider the worst-case input of ± 5. Conversely, in our experiments, we find that the distribution of inputs follows an approximately Laplace distribution as shown in Appendix C. Thus, we observe a much smaller empirical failure probability.

While our default parameters were experimentally chosen to give high classification accuracy (Section 6, shows a minor degradation over plaintext accuracy), it is unclear if these values are optimal. For example, one approach to reducing the failure probability is to decrease p from 16 to 10-bit (which gives a failure probability of 2^{-28}). However, the trade-off is that the intermediate x_i' values will be truncated severely. For instance, with $n = 4$, x_i' will be truncated to 5-bits. We find that this loss of precision is too significant to simulate a

ReLU function accurately. Similarly, while a higher degree polynomial might better approximate a ReLU, a higher degree will negatively affect both the failure probability and the precision loss of x'_i. Future work could conduct an extensive hyperparameter search of all parameters to find the optimal trade-off. However, as we see in Section 6 (Tables 2-5), this would *at most* yield a 0.5% increase in encrypted classification accuracy (PyTorch vs. CrypTen accuracy), for the worst model and dataset.

Evaluating Algorithm 2. In Figure 2, we plot this approach alongside the previous approaches to evaluate the runtime. ESPN incurs slightly more overhead in the LAN setting; however, it scales significantly better (the confidence intervals do not overlap) to wide area networks that can be expected in practice.

4.4 Alternative Single Round Protocol: HoneyBadger

Like ESPN, Lu et al. give a single round protocol for exponentiation in MPC [28]. Despite focusing on a completely different problem (anonymous communication), they provide an MPC protocol of independent interest for exponentiation, which we also utilize in our work. They take a very different approach to our work that yields different trade-offs. Instead of the binomial theorem, their work utilizes the following factoring rule

$$x^k - r^k = (x-r)\sum_{i=0}^{k-1} x^{k-i-1} r^i \tag{4}$$

where r is a random secret-shared number derived during pre-computation. We assume each party has a share of x and a share of r^i for $i \in \{1,\ldots,k\}$ before beginning the protocol (instead of the more common Beaver triplets). The first step in the protocol is to compute and reveal $x-r$ (x blinded by r), which uses a single round. Once revealed, this value becomes a public constant C. After some algebraic manipulation of (4), Lu et al. obtain a recursive formula for $x^i r^j$ given below.

$$[[x^k r^j]] = [[r^{k+j}]] + C\sum_{i=0}^{k-1} [[x^{k-i-1} r^{i+j}]] \tag{5}$$

Using dynamic programming, the parties can then compute any power ($x^k r^0$) using only additions of previously computed terms and powers of r. To compute polynomials using this protocol, we simply swap the call to *Exp* in line 6 of Algorithm 2.

The advantage of Lu et al.'s protocol is that the communication is small (only the opening of $x-r$). The primary disadvantage is that the protocol requires a modified pre-computation phase, which is as difficult to pre-compute securely as the original problem (it is exponentiation). On the contrary, our binomial protocol uses standard Beaver triplets commonly found in MPC frameworks. There are well established protocols for efficiently computing these triplets, and the parties may already have them due to the popularity of Beaver's trick. A more minor disadvantage of Lu et al.'s solution is that, while the protocol requires very little communication, it is not locally parallelizable as each dynamic programming step depends on the previous one. In contrast, our entire protocol can be executed in parallel.

We also consider the runtime of using HoneyBadger in Figure 2. We emphasize this is a runtime-only evaluation. Without our training algorithm in Section 5, none of the polynomial-based solutions can attain usable accuracy. We find that ESPN and HoneyBadger perform similarly in practice, with HoneyBadger gaining a slight advantage in very low network delay. Due to the pre-computation trade-offs, we will evaluate both approaches for the remainder of this work.

5 PILLAR: Polynomial Activation Regularization

Our initial benchmark in Section 4 showed a significant speedup when replacing ReLU functions with polynomials implemented using ESPN and HoneyBadger. However, a notable challenge neglected thus far is that replacing a ReLU with a polynomial can drastically reduce the accuracy of the model [12, 18]. This section discusses the causes of the accuracy degradation and describes our mitigation techniques. Finally, we give empirical results showcasing the high accuracy of our modified training procedures across various architectures and datasets.

5.1 The Problem with Polynomial Activation Functions

Escaping Activations. The first step in replacing an activation function with a polynomial is to design a polynomial that closely approximates the original function. A common approach for this is the least-squares polynomial fitting. In this approach, a table of values is created for the polynomial over a small discretized range of values. This creates a system of equations for the polynomial coefficients that can be solved with least squares. The challenge with this approach is that, outside of this range, the polynomial no longer resembles the original activation function and often diverges rapidly. This leads to a problem called escaping activations, first identified by Garimella et al. [12]. If one naively swaps a ReLU for its polynomial approximation, all weights will become infinite within a few training epochs. We give an example of this degradation in Figure 3. We can see that, without modifying the training procedure, a polynomial can completely destroy the accuracy.

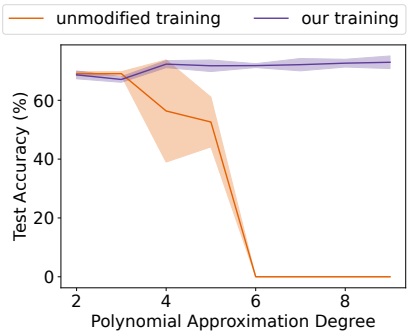


Figure 3: Accuracy of a 2-layer convolutional network trained with varying degrees for the polynomial activation function.

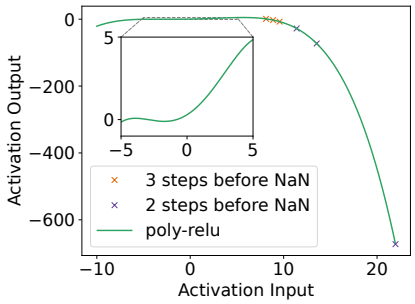


Figure 4: Illustrating the escaping activation problem for the two layers convolutional network.

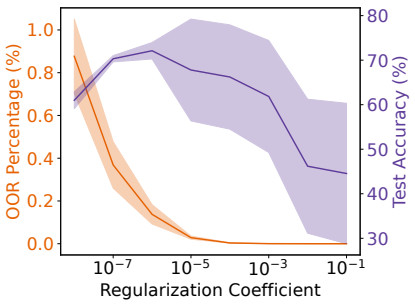


Figure 5: The effect of the regularization coefficient, β on model accuracy and out-of-range ratio for $\gamma = 10$

To illustrate the problem more clearly, we conduct an experiment using a polynomial of degree four fitted on the range $[-5,5]$ (λ=5) as the activation function for a three-layer model on CIFAR-10 [23]. In Figure 4, we plot this polynomial activation function and the ℓ_∞-norm of the input and output to each activation function. We note that, with no modification (except replacing ReLUs with polynomials), the weights of this model become undefined within approximately three epochs of training. First, we note the divergent behaviour of the polynomial outside the fitted range. Second, we observe the effect of the divergence on the outputs of the activation function. Specifically, we wait until the model weights become undefined (NaN in Python) and then observe the behaviour leading up to the explosion. We can see that three steps before the model weights become undefined (NaN), the input values of each activation are out-of-range, but the outputs still behave similarly to a ReLU. However, in the next iteration (two steps before NaN), a single value in the first layer goes too far out of range. This causes a ripple effect for the other two layers, creating an extremely large output (approx -2000) in the final activation function. This large value creates a large gradient, and after another iteration of training, the values become so large that the gradients (and weights) become undefined (NaN). We find that minimizing the classification loss alone is not enough to keep the model in range as the gradients explode before decreasing the loss.

Truncated Polynomial Coefficients. An additional challenge is that we will evaluate the fitted polynomial in a finite ring with limited precision. This significantly impacts the polynomial coefficients, which tend to be relatively small, especially for the higher-order terms. Specifically, these small coefficients can get truncated to zero in limited precision, which causes the polynomial to diverge even inside the fitted range. We give an example of this in Appendix C.

5.2 Defining PILLAR

Our approach, which we call PILLAR, is the combination of the components we describe in this section. Activation function regularization is our primary approach for mitigating escaping activation functions. However, to scale to larger models, we find that the additional steps of clipping, regularization warm-up, and adding batch normalization are beneficial.

Quantization-Aware Polynomial Fitting. We begin by solving the problem of truncated polynomial coefficients. To address this, we fit the polynomial with the precision constraint in mind. We do this by using mixed integer non-linear programming. Let X be the set of all values between $[-\lambda,\lambda]$ in p-bit precision (the domain we want to fit on). First, we generate $Y = ReLU(X) \cdot 2^p$, a table of values for a standard ReLU scaled up by the precision. Scaling the output of the ReLU allows us to work in the integer domain (similar to fixed point arithmetic). We then compute a matrix B where each column is the different powers of X used in a polynomial ($B = [X^0, X^1, X^2, \ldots]$).

Next, we solve the system $AB = Y$ for A using mixed integer linear programming with $A \in [-2^p - 1, 2^p - 1]$ to get the coefficients A that minimize the error between the polynomial AB and the ReLU values Y. Finally, we scale the resulting coefficients down by 2^p. We note that $A \in [-2^p - 1, 2^p - 1]$ corresponds to coefficients being bounded by $[-1, 1]$ after we scale down. We empirically choose $p = 10$ for all polynomials during the ML training. As we observe in Appendix C, our quantized polynomial fitting addresses the problems of exploding activations within the range. However, the issue of going out-of-range requires additional treatment.

Activation Regularization. Following the observations of Section 5.1 and Garimella et al. [12], it is clear that minimizing the classification loss alone is not sufficient to pre-

vent escaping activations. Garimella et al. proposed QuaIL, a method that trains one layer of the model at a time, focusing not on classification accuracy but the similarity of the layer to a standard ReLU model [12]. QuaIL showed much better accuracy than naive training but only scaled to models with at most 11 layers.

In our work, we address the cause of the problem directly by regularizing the input to each activation function during training. We add an exponential penalty to the loss function when the model inputs out-of-range values to the polynomial activation function. Let x be the input to the activation function, and λ_{reg} be the upper bound of the symmetric range $[-\lambda_{reg}, \lambda_{reg}]$ in which we would like the input to be contained. Then, we define our penalty function as

$$p(x) = \left(\frac{x}{\lambda_{reg}}\right)^{\gamma} \tag{6}$$

where γ is a large even number (to handle negative values) determining the severity of the penalty. We find that values between six and ten work best in practice, with $\gamma = 10$ being the default in our experiments. This penalty function gives negligible penalties (less than 1) for $|x| < \lambda_{reg}$ and rapidly grows (in the degree of γ) as $|x| > \lambda_{reg}$.

We aggregate $p(x)$ over I, the set of inputs to all activation functions, by taking the average over each activation layer in the model. After aggregation, we scale the penalty using a regularization coefficient β and add it to the existing cross-entropy loss function of the model ℓ_c. Specifically, the modified loss function ℓ' is defined as:

$$\ell'(\cdot) = \ell_c(\cdot) + \frac{\beta}{K} \sum_{x \in I} p(x) \tag{7}$$

where K is the number of activation layers in the model. This allows us to tune the importance of classification loss vs. the cost of going out-of-range.

Clipping. Although activation regularization teaches the model not to go out-of-range over time, the model still needs to avoid going to infinity during the early stages of training. Thus, during training, we apply a clipping function to the input of the activation function such that if any input goes out of range, it is truncated to the range's maximum (or minimum) value. This clipping function does not affect the penalty as it is applied after the penalty function has been computed. We emphasize that this clipping function is only used during training and is removed during inference. The intuition is that the model should learn not to go out-of-range during training and thus no longer requires this clipping function during inference. Additionally, we find that setting the λ_{reg} of the penalty to be smaller than the range used for polynomial fitting (and clipping) can yield even better results. This is because the polynomial will be accurate for a larger range outside of the range the model was regularized to stay inside, allowing an extra buffer in case of failure during inference.

Dataset	Model	Plain Accuracy ± CI	
		ReLU	PILLAR
Cifar10	MiniONN	91.2 ± 0.17	88.1 ± 0.26
	VGG 16	92.6 ± 0.16	90.8 ± 0.11
	ResNet18	94.7 ± 0.09	93.4 ± 0.14
	ResNet110	92.8 ± 0.27	91.4 ± 0.18
CIFAR-100	VGG 16	70.9 ± 0.17	66.3 ± 0.22
	ResNet32	68.4 ± 0.46	67.8 ± 0.32
	ResNet18	76.6 ± 0.07	74.9 ± 0.14
ImageNet	ResNet50	80.8	77.7

Table 1: Plain-text Accuracy of PILLAR (5 runs).

Regularization Warm-up. We find the minimum requirements for successfully training a model with polynomial activation functions are activation regularization and clipping. However, for larger models, the penalty term can be extremely large in the first few epochs (until the model learns to stay in range). In some cases, the loss can become infinite due to our regularization penalty. To address this challenge, we adopt a regularization scheduler for the first four epochs that slowly increases both γ and β to the values used for the rest of the training. Empirically, the following schedule works well and avoids infinite loss. We let $\gamma' \in \{4, 6, \ldots, \gamma, \gamma, \ldots\}$ and $\beta' \in \{\beta/100, \beta/50, \beta/10, \beta/5, \beta, \beta, \ldots\}$.

BatchNorm Layers. Garimella et al. also investigated using normalization to help prevent the escaping activation functions [12]. They proposed a min-max normalization approach where each layer's minimum and maximum values are approximated using a weighted moving average of the true minimum and maximum. These values are frozen during inference. Garimella et al. observed that this approach alone was insufficient, as activations still escaped the range during inference. We observe this operation is similar to the batch norm layer commonly added to ML models. The main difference is that the mean and standard deviation of the batch are used to normalize the layer instead of the minimum and maximum values. By fixing the approximation of the mean and standard deviation during inference (following CrypTen [22]), this operation is very efficient in MPC. We study the effect of BatchNorm in Appendix A. We find that batch norm layers considerably improve the accuracy of PILLAR. This is an intuitive result as batch normalization helps to keep each layer's output bounded and thus reduces the work of our regularization function.

Summary of PILLAR We refer to PILLARas the combination of all components described in this section. We note that the clipping and regularization warm-up components are used (only during training) to enable regularization by preventing the model from going to infinity. The regularization

component ensures that the model will stay in range during inference. All three of these components play a crucial role in the success of PILLAR, as removing any of them will result in a model with unacceptable accuracy (due to infinite weights or escaping activations). The batch norm is the only optional component, which we give a small ablation study over in Appendix A.

5.3 Measuring PILLAR's Effectiveness

The Regularization Coefficient. To show the effect of our regularization and coefficient β, we conduct an experiment using the same three-layer model on CIFAR-10 from Section 5.1. In Figure 5, the left y-axis gives the out-of-range ratio (OOR), defined as the ratio of activation function inputs that were not within the interval $[-5,5]$. The right y-axis is standard classification accuracy, and the x-axis varies the regularization coefficient β. We observe that when the coefficient, β, is small, the model goes out-of-range often and thus has poor accuracy. As we increase β, the out-of-range ratio decreases, and accuracy increases. However, if we increase the coefficient too much, the accuracy decreases again.

End-to-end Accuracies. We evaluate PILLAR across a range of different models and architectures considered in related work [18,31,35]. We summarize the results in Table 1. We include the accuracy of a model trained with standard ReLUs as a baseline. All results are averaged over five random seeds, and we show the 95% confidence interval. The only exception is ResNet50 on ImageNet, where we only train a single model due to the size of the dataset. We defer to Section 6 for the details of the experimental setup. We note that these results are using PyTorch with no cryptography or quantization. We give a complete evaluation using MPC in Section 6 where quantization has an effect. Table 1 provides preliminary evidence that our polynomial training approach yields high accuracies competitive with state-of-the-art ReLU models across a range of models and datasets.

6 Evaluation of Co-Design

In this section, we provide an end-to-end comparison of our co-design against state-of-the-art solutions in secure inference. We evaluate the performance of PILLAR and Algorithm 2 using both ESPN and HoneyBadger as they offer different trade-offs in the type of pre-computation needed and the size of the communication. We determine the state-of-the-art works following a recent SoK by Ng and Chow [32]. Specifically, we consider three solutions on the Pareto front of latency and accuracy as determined by Ng and Chow. These works are COINN [18], GForce [31] and CrypTen (CryptGPU) [22,35]. We also evaluate Cheetah [17], a recent work not included in the SoK. We will evaluate the metrics of latency (or runtime of a single sample) and encrypted accuracy. We additionally evaluate the communication and number of rounds in Appendix D. We begin with the experimental setup, then evaluate both metrics (runtime and accuracy) against each related work. Section 6.2 evaluates the ResNet-18 architecture, which gives our state-of-the-art performance. Section 6.3 evaluates the VGG-16 architecture, the only architecture GForce evaluates. Section 6.4 considers other ResNets and the MiniONN architecture following COINN. Finally, in Section 6.5, we evaluate ImageNet against Cheetah, COINN, and CryptGPU.

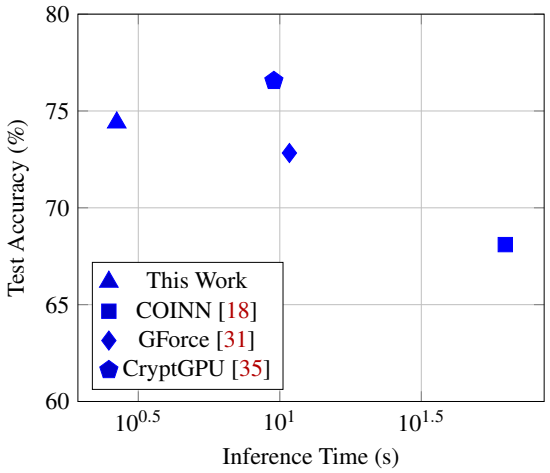


Figure 6: Summary of the inference time vs accuracy for each state-of-the-art approach on the CIFAR-100 dataset in the WAN (100 ms roundtrip delay).

Results Summary. We plot a summary of the accuracy and inference time for CIFAR-100 in Figure 6. For both datasets (recall Figure 1), we observe that our work always gives the solution with the fastest inference time by a statistically significant amount. In terms of accuracy, our work is competitive with the state-of-the-art, but CryptGPU is always the most accurate as it can infer unmodified plaintext models. Our solution is faster than CryptGPU by 4×, GForce by 5×, and COINN by 18× on average in wide area networks. Our accuracies are competitive with state-of-the-art and plaintext solutions and stay stable (no escaping activations) with models containing up to 110 layers and 23 million parameters.

6.1 Experimental Setup

We develop an experimental setup that follows as closely as possible to the works we compare to [18, 31, 35]. We use CIFAR-10/100 [23] and ImageNet [10], the same common benchmark datasets as related work. Our model architectures include: MiniONN [26], VGG [34], and ResNets [16]. This covers models of depth 7 to 110 layers with the number of trainable parameters ranging from 0.2 to 23 million.

Implementation Details. All experiments are run on a machine with 32 CPU cores @ 3.7 GHz and 1 TB of RAM with two NVIDIA A100 with 80 GB of memory. We simulate network delay by calling the sleep function for the appropriate time whenever the client and server communicate. We simulate the LAN with 0.25 ms roundtrip delay and the WAN with 100 ms, following COINN [18]. We additionally evaluate a real WAN using AWS instances in Section 6.6. All experiments (except ImageNet) are repeated over multiple random seeds, and we report the mean and 95% confidence interval as shaded areas.

For all related work, we run our own benchmarks of their code unmodified. While results for ResNet32 and MiniONN appear in the Cheetah paper, there was no source code for these models so we only evaluate Cheetah on ImageNet. To use CryptGPU in practice, one must first train a model in PyTorch. For ImageNet, PyTorch provides pre-trained models. However, we will need to train a model for all other architectures and datasets. We simply use the same configurations as our PolyRelu models but with standard ReLUs. We include all source code to reproduce our results [1].

Hyperparameters. We introduce five new hyperparameters associated with our techniques: polynomial degree (n), polynomial approximation range (λ), polynomial regularization range (λ_{reg}), polynomial regularization coefficient (β), and polynomial regularization exponent (γ). The default values for each are decided by extensive grid searches. These parameters primarily affect the accuracy and not the inference time, except for the polynomial degree (n), which has a minor effect on the communication size (but not the rounds). We found the optimal quantization-aware polynomial degree (n) to be 4. We found that higher degrees than four increase the failure probability of truncation with minimal accuracy gains (Section 4.3). Conversely, lower degrees give lower accuracy (as shown in Figure 3). The polynomial used in all evaluations is: $0.31445312 + 0.5x + 0.15625x^2 - 0.00292969x^4$.

For all models and datasets, we found a value of $\gamma = 10$ performs well as it introduces a strong enough incentive for PolyReLU inputs to stay in range (smaller values did not) while keeping penalization for values in range practically 0 (if an input x is within range, then $\frac{x}{\lambda_{reg}} < 1 \Rightarrow (\frac{x}{(\lambda_{reg})^{10}} \sim 0)$. Larger values than this often lead to an infinite penalty term. We found that a polynomial approximation range $\lambda = 5$ provides a good compromise between regularization and quantization. We found that smaller values of λ destroy the accuracy during training, and larger values use too much precision, increasing the failure probability. The optimal polynomial regularization range (λ_{reg}) and polynomial regularization coefficient (β) vary per model and dataset, although we found $\lambda_{reg} = 4.8$ (slightly tighter than $\lambda = 5$) and $\beta = 5 \times 10^{-5}$ to be good default values. We use $p = 10$ during ML training, but revert to CrypTen's default precision of $p = 16$ during inference in a 64-bit ring.

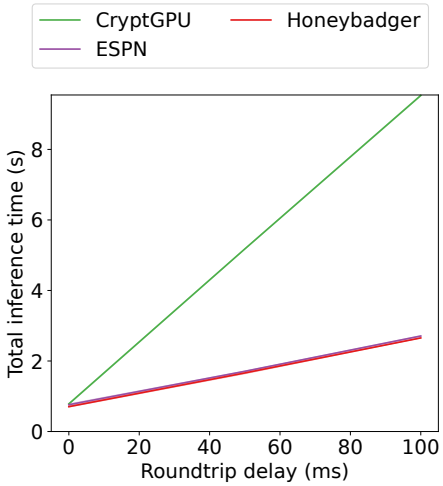


Figure 7: ResNet-18 evaluation on CIFAR-100 (20 runs).

We used Stochastic Gradient Decent as the optimizer with a learning rate of 0.013 as the default. This includes a Cosine Annealing Learning Rate Scheduler with Linear Learning Rate Warmup of 5 epochs and decay 0.01. We use a weight decay of either 10^{-4} or 5×10^{-4} and a momentum of 0.9. We used a default batch size of 128 and set the default number of Epochs to 185. For some models, we tuned the learning rate, number of epochs, and regularization coefficient to achieve a slightly higher accuracy. We detail hyperparameters in our source code repository.

6.2 ResNet-18 Architecture

In this section, we use a ResNet-18 architecture as it is the architecture that yields the best inference time and accuracy over all CIFAR-10 and CIFAR-100 experiments. For this comparison we focus on CryptGPU, which has been shown to be a state-of-the-art solution [32]. CryptGPU (or CrypTen) [35] serves as a baseline in all our comparisons including those against GForce and COINN in Sections 6.3 and 6.4. Neither COINN nor GForce support the ResNet-18 architecture evaluated in this section.

Inference Time. We measure the inference time of a single input image over varying network delays. The results are given in Figure 7. We include the result for CIFAR-100 and omit the plot for CIFAR-10 as it displays similar trends. We observe that both PILLAR + HoneyBadger and PILLAR + ESPN outperform CryptGPU with statistical significance across all roundtrip delays (as the shaded area does not overlap). In the WAN (100 ms), this corresponds to a 4× speedup over CryptGPU. Furthermore, we find HoneyBadger and ESPN perform similarly as observed in Section 4, with PILLAR + HoneyBadger having a slight advantage.

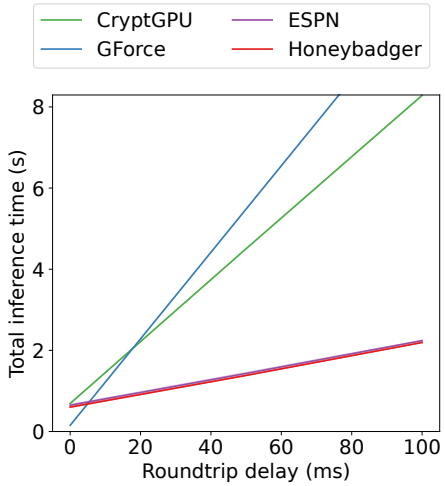


Figure 8: VGG-16 evaluation on CIFAR-100 (20 runs).

Accuracy. We measure the accuracy of the models on the testing set both in plain (using PyTorch) and encrypted (using CrypTen). We give the result in Table 2. First, we observe the plain and encrypted accuracies are very similar, indicating that quantization has a minor effect despite not considering this in training. We find that CryptGPU and PILLAR give similar accuracies, with CryptGPU performing slightly better, as is to be expected since they use unmodified activation functions. However, we argue this slight loss in accuracy is well justified by the significant decrease in inference time.

Dataset	Technique	Plain Acc	Enc Acc
CIFAR-10	PILLAR	93.4 ± 0.14	93.3 ± 0.22
	CryptGPU	94.7 ± 0.09	94.6 ± 0.10
CIFAR-100	PILLAR	74.9 ± 0.14	74.4 ± 0.31
	CryptGPU	76.6 ± 0.07	76.6 ± 0.13

Table 2: ResNet-18 accuracy comparison (5 runs).

6.3 VGG-16 Architecture

In this section, we compare with GForce, the current state-of-the-art as shown by Ng and Chow [32]. GForce focused on a modified VGG-16 [34] architecture and compared it to all other works (including those using different architectures). For completeness, we evaluate the VGG-16 architecture using our techniques, and CryptGPU although we note that the ResNet-18 architecture outperforms VGG-16 in both inference time and accuracy. COINN [18] does not give results for VGG-16, so we exclude it from this section.

Inference Time. Since our work aims to reduce the rounds needed by binary non-linear layers, we replace the MaxPool layers in the VGG-16 with AvgPool for all solutions (including CryptGPU and GForce). We give the inference times over various delays in Figure 8. First, we note that GForce significantly outperforms all other solutions in the LAN. However, for more realistic high latency networks (>5ms roundtrip delay), we observe our solutions significantly outperform GForce (5× speedup in WAN). Once again, our solutions outperform CryptGPU for all network delays.

Encrypted Accuracy. We recall that we swap the MaxPool layers for AvgPool layers in the inference time evaluation. This comes at a cost to accuracy for the VGG architecture. Thus, to give the best scenario possible for GForce, we consider the accuracy of GForce with MaxPools and our work with AvgPool. We give the results in Table 3. As expected, GForce outperforms our work in accuracy (due to the MaxPools); however, only by a few percentage points. We train CryptGPU to use AvgPool and find it also loses a few percentage points, confirming our hypothesis that a MaxPool is necessary for high accuracy in VGG-16. We emphasize that our ResNet-18 result outperforms GForce's VGG result in inference time and accuracy. Furthermore, ResNets are a more popular and compact architecture due to skip-connections.

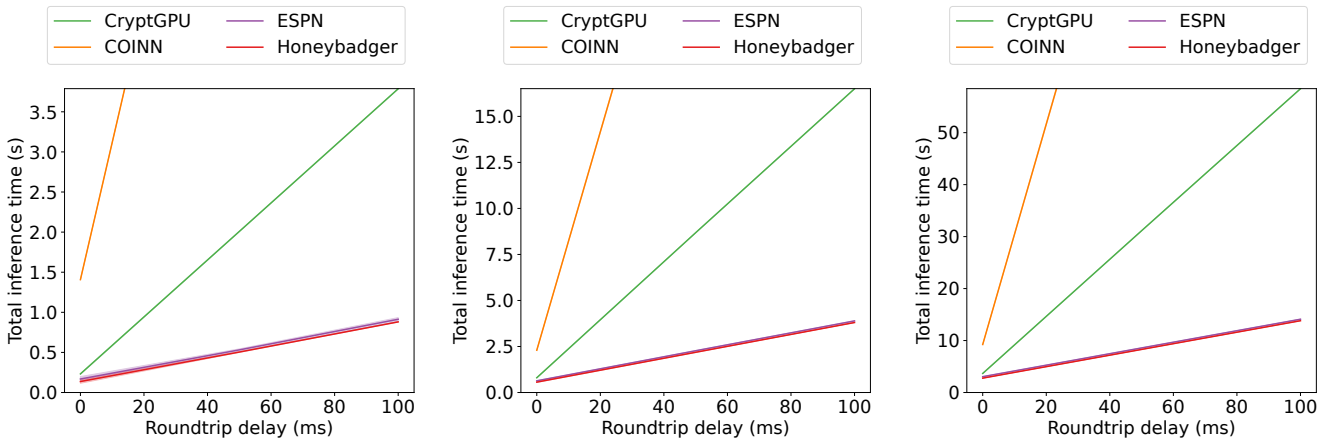
Dataset	Technique	Plain Acc	Enc Acc
CIFAR-10	PILLAR	90.8 ± 0.11	90.8 ± 0.14
	CryptGPU	92.6 ± 0.16	92.5 ± 0.16
	GForce	-	93.12
CIFAR-100	PILLAR	66.3 ± 0.22	66.3 ± 0.32
	CryptGPU	70.9 ± 0.17	70.8 ± 0.13
	GForce	-	72.83

Table 3: VGG-16 accuracy comparison (5 runs).

6.4 Other Architectures

While GForce is the current state-of-the-art, COINN is a competitive solution that evaluates ResNet architectures. Thus, we also evaluate the same configurations as COINN. This includes the smaller MiniONN architecture, a ResNet-32, and a ResNet-110. We exclude GForce from this evaluation as they only evaluate VGG-16 models.

Inference Time. We again swap all MaxPool layers for AvgPool in our work, and CryptGPU but leave COINN unmodified. We give the results in Figure 9. We observe that, over each of the three increasingly large architectures, the trends are similar and proportional to the number of parameters (0.2, 0.5, and 1.7 million parameters for MiniONN, ResNet-32, and ResNet-110, respectively). Across all architectures and network delays, our work outperforms COINN by a statistically significant amount (18× on average in WAN). We once



(a) MiniONN on CIFAR-10 (b) ResNet-32 on CIFAR-100 (c) ResNet-110 on CIFAR-10

Figure 9: Evaluating the various COINN architectures (20 runs).

again outperform CryptGPU in all evaluations[1] with a 4× speed up on average in the WAN.

Encrypted Accuracy. We give the results in Table 4. We observe that PILLAR is competitive with related work in all models, although we remark that, once again, our ResNet-18 models outperform all others. We also note that, while COINN does quantization-aware training, PILLAR does not and still only loses a small amount of accuracy in encryption vs. plaintext.

Dataset/Model	Technique	Plain Acc	Enc Acc
CIFAR-10 / MiniONN	PILLAR	88.1 ± 0.26	87.9 ± 0.46
	CryptGPU	91.2 ± 0.17	91.2 ± 0.16
	COINN	-	87.6
CIFAR-10 / ResNet-110	PILLAR	91.4 ± 0.18	91.4 ± 0.25
	CryptGPU	92.8 ± 0.27	92.7 ± 0.26
	COINN	-	93.4
CIFAR-100 / ResNet-32	PILLAR	67.8 ± 0.32	67.8 ± 0.47
	CryptGPU	68.4 ± 0.46	68.5 ± 0.45
	COINN	-	68.1

Table 4: Accuracy comparison on the various architectures considered in COINN (5 runs).

6.5 Scaling to ImageNet

In this section, we evaluate the scalability of our approach on the ImageNet dataset using a ResNet-50 architecture with 23 million parameters. This architecture was previously too large for training with polynomial activation functions [12]. We compare our approach to Cheetah, COINN and CryptGPUand exclude GForce as they do not consider ImageNet.

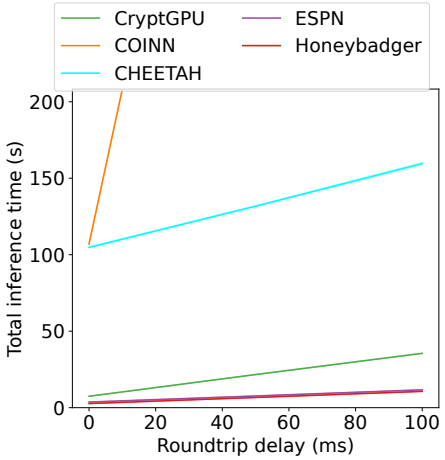


Figure 10: ImageNet evaluation on ResNet-50 (20 runs).

Inference Time. We plot the inference time in Figure 10. We observe a significant reduction over Cheetah and COINN across all network delays. We outperform Cheetah by 39× in the LAN (0.25 ms) and 15× in the WAN (100 ms). Over COINN, we observe a 28× reduction in the LAN and a 90× reduction in the WAN. Compared to CryptGPU we find that PILLAR + HoneyBadger is the fastest in all network delays by 3× on average. PILLAR + ESPN is slightly slower in the LAN, but once again outperforms CryptGPU in the WAN.

[1] All of which are statistically significant except MiniONN in LAN where the confidence intervals overlap slightly.

Dataset / Model	ESPN	HoneyBadger	CryptGPU
CIFAR-10 / ResNet-110	49.3 ± 0.1	49.0 ± 0.1	242.5 ± 0.9
CIFAR-10 / ResNet-18	15.3 ± 0.2	12.9 ± 0.1	48.9 ± 0.1
CIFAR-100 / ResNet-18	15.4 ± 0.2	12.9 ± 0.1	48.9 ± 0.5
CIFAR-100 / ResNet-32	14.1 ± 0.1	14.0 ± 0.1	75.4 ± 1.6
ImageNet / ResNet-50	153.9 ± 1.0	104.9 ± 0.8	268.67 ± 1.3

Table 6: Real World WAN evaluation. We report total inference time in seconds.

Encrypted Accuracy. We present a summary of the accuracies in Table 5. We note that Cheetah [17] did not measure accuracy in either their code or paper so we omit them from this comparison. We observe a much higher encrypted accuracy for PILLAR compared to COINN and thus, our solution is Pareto dominant. For CryptGPU, we use a pre-trained PyTorch model with state-of-the-art accuracy. Therefore, as expected, CryptGPU has an accuracy 3% higher than the model we trained from scratch. We note that with a higher degree polynomial, we were able to train a 79.2% polynomial model. However, this model is not possible to infer in the 64-bit field used by CrypTen (as higher degrees need more precision by equation 10). We discuss future directions to further improve this result in Section 7.

Technique	Plain Acc	Enc Acc
PILLAR	77.7	77.3
CryptGPU	80.8	80.8
COINN	-	73.9

Table 5: ImageNet accuracy comparison (1 run).

6.6 Real WAN Evaluation

This section gives benchmarks for ESPN, HoneyBadger, and CryptGPU [35] in a real WAN. We use two AWS EC2 `g4dn.metal` instances, one in the Ohio data centre and one in Frankfurt, Germany. Each machine has 96 cores, 384 GB of memory, 100 GB/s network bandwidth, and a NVIDIA T4 GPU. In practice, we measured 10.8MB/s bandwidth between the two instances. We run all the ResNet architectures and summarize the results in Table 6. We repeat each experiment 20 times and report the mean and 95% confidence intervals. We observe similar trends to the simulated WAN used in previous experiments. Namely, ESPN and HoneyBadger perform similarly in runtime for most models. The exception is larger models like ResNet50, where the bandwidth limits impact ESPN more than HoneyBadger. In all cases, CryptGPU is significantly outperformed by both approaches.

7 Discussion

Our experimental evaluation in Section 6 showed our algorithms significantly outperform all related work in WAN inference time. While state-of-the-art compared to other polynomial training approaches, PILLAR still incurs a minor accuracy degradation compared to standard models with ReLUs. We posit a few directions for future work to further close this gap between polynomials and ReLUs.

Quantization. Note that aside from our quantization-aware polynomial fitting described in Section 5, we have made no efforts to reduce the effects of quantization. COINN developed training algorithms to help the model be robust to the overflow and quantization present in MPC [18]. An interesting future work would be to combine the COINN methods with PILLAR to see if further accuracy gains are possible.

Precision. By using CrypTen as our backend, we were limited to a 64-bit ring for cryptographic operations. As discussed in Section 4.3, this precision determines the degree and range of polynomials we can use (due to either the failure probability of truncation or severe truncation of intermediate values). Interesting future work is to increase this precision to enable higher-degree polynomials and study the performance-accuracy trade-off. Our initial results on ImageNet show that we can train up to a degree eight polynomial without suffering escaping activations. However, we could not increase the ring size to study the effect of higher degrees on inference time.

MaxPools. We recall that a MaxPool layer requires comparisons and, thus, expensive conversions to binary shares (like ReLUs). Therefore, we replaced all MaxPools with AvgPool layers. However, in some architectures, such as VGG-16 [34], we found that swapping MaxPool for AvgPool degraded accuracy by up to 6%. Finding an efficient MaxPool alternative for architectures like VGG is important for future work. However, since the ResNet models give high accuracy using AvgPool layers we did not pursue this issue further.

8 Related Work

This work focuses on achieving state-of-the-art run time and accuracy in two-party secure inference. We measure this objective by evaluating against the current state-of-the-art as determined by a recent SoK by Ng and Chow [32]. Namely, we compare to COINN [18], GForce [31] and CrypTen [22, 35] in Section 6 as they represent the Pareto front according to Ng and Chow [32]. Another potential candidate on the Pareto front is Falcon [25], with low latency and accuracy [25]. We did not evaluate Falcon as the accuracy drop was too significant (over 10% [32]). Furthermore, GForce is shown to

outperform Falcon in both latency and accuracy, and we outperform GForce [31]. For a complete list of other works not on the Pareto front, we defer to Ng and Chow's work [32]. Notably, many works consider different threat models or use different approaches, such as homomorphic encryption. We leave extending our polynomial activation functions to these settings for future work. For the remainder of this section, we discuss works with a similar approach to ours that are not state-of-the-art or not evaluated by Ng and Chow [32].

Replacing or Reducing ReLU's. It has been established that the non-linear functions such as ReLU are the bottleneck for secure computation [12, 13, 18, 29]. Several works initially focused on reducing the number of ReLU activations, optimizing for the best trade-off between accuracy and runtime [13, 20]. A faster approach is to replace all ReLU's entirely using polynomial approximations [12]. In Section 5, we discussed the most recent work in this space, Sisyphus [12]. While making significant progress toward training models with polynomial activations, Sisyphus could not overcome the escaping activation problem for models with more than 11 layers. Before Sisyphus, there were a handful of works on smaller models that typically focus on partial replacement (some ReLU's remained) [14, 29, 30]. An interesting exception from Lee et al. used degree 29 polynomials in HE but suffered prohibitively high runtimes [24]. Our work is the first to make high-accuracy polynomial training feasible (without escaping activations) in deep neural networks.

A notable recent work is PolyKervNets [3]. Inspired by the computer vision literature, PolyKervNets remove the activation functions and instead exponentiate the output of each convolutional layer [3]. The problem with this approach is that, similar to polynomial activation functions, the exponents make the training unstable. Aremu and Nandakumar note that exploding gradients prevent their approach from scaling to ResNet models deeper than ResNet18 (using degree 2 polynomials). Furthermore, PolyKervNets only allow for a single fully connected layer which reduces the accuracy of the models. Conversely, PILLAR scales to deeper models such as ResNet110 and much higher degrees. Moreover, we achieve significantly better plaintext accuracy on ResNet-18 (93.4 vs 90.1 on CIFAR-10 and 74.9 vs 71.3 on CIFAR-100).

Polynomial Evaluation in MPC. Our work focuses on co-designing the activation functions with cryptography by using polynomials. However, the problem of computing polynomials in MPC is of independent interest and has also been studied in the literature. The state-of-the-art in this space is HoneyBadger, as discussed in Section 4. Other notable works include the initial inspiration for HoneyBadger from Damgård et al. [8]. This early approach conducts exponentiation by blinding and reconstructing the number to be exponentiated so the powers can be computed in plaintext [8]. Building off this idea, Polymath constructs a constant round protocol for evaluating polynomials focused on matrices [27]. However, HoneyBadger outperforms Polymath by reducing both the rounds and the number of reconstructions to one.

9 Conclusion

In this work, we co-designed the ML and MPC aspects of secure inference to remove the bottleneck of non-linear layers. PILLAR maintains a competitive inference accuracy while being significantly faster in wide area networks using novel single round MPC protocols (ESPN and HoneyBadger). Our state-of-the-art inference times motivate future work to further improve the ML accuracy of polynomial activations in DNNs.

Acknowledgments

We gratefully acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) for grants RGPIN-05849, and IRC-537591, the Royal Bank of Canada, and Amazon Web Services Canada.

Availability

We make all source code to reproduce our experiments available here: https://github.com/LucasFenaux/PILLAR-ESPN.

References

[1] https://github.com/LucasFenaux/PILLAR-ESPN.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA, October 2016. Association for Computing Machinery.

[3] Toluwani Aremu and Karthik Nandakumar. PolyKervNets: Activation-free Neural Networks For Efficient Private Inference. In *First IEEE Conference on Secure and Trustworthy Machine Learning*, February 2023.

[4] Donald Beaver. Foundations of secure interactive computing. In *Advances in Cryptology — CRYPTO 1991*, pages 377–391, 1991.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, January 1988. Association for Computing Machinery.

[6] Octavian Catrina. Round-Efficient Protocols for Secure Multiparty Fixed-Point Arithmetic. In *2018 International Conference on Communications (COMM)*, pages 431–436, June 2018.

[7] Octavian Catrina and Sebastiaan de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks*, Lecture Notes in Computer Science, pages 182–199, Berlin, Heidelberg, 2010. Springer.

[8] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 285–304, Berlin, Heidelberg, 2006. Springer.

[9] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Proceedings 2015 Network and Distributed System Security Symposium*, San Diego, CA, 2015. Internet Society.

[10] 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, June 2009.

[11] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, Lecture Notes in Computer Science, pages 823–852, Cham, 2020. Springer International Publishing.

[12] Karthik Garimella, Nandan Kumar Jha, and Brandon Reagen. Sisyphus: A Cautionary Tale of Using Low-Degree Polynomial Activations in Privacy-Preserving Deep Learning, November 2021.

[13] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. CryptoNAS | Proceedings of the 34th International Conference on Neural Information Processing Systems. *Advances in Neural Information Processing Systems*, 33:16961–16971, 2020.

[14] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210. PMLR, June 2016.

[15] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge university press, 2009.

[16] 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.

[17] 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, Boston, MA, August 2022. USENIX Association.

[18] 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*, CCS '21, pages 3266–3281, New York, NY, USA, November 2021. Association for Computing Machinery.

[19] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, SEC'20, pages 1345–1362, USA, August 2020.

[20] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. DeepReDuce: ReLU Reduction for Fast Private Inference. In *Proceedings of the 38th International Conference on Machine Learning*, pages 4839–4849. PMLR, July 2021.

[21] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 830–842, New York, NY, USA, 2016. Association for Computing Machinery.

[22] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.

[23] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[24] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data, June 2021.

[25] Shaohua Li, Kaiping Xue, Bin Zhu, Chenkai Ding, Xindi Gao, David Wei, and Tao Wan. FALCON: A Fourier Transform Based Approach for Fast and Secure Convolutional Neural Network Predictions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8705–8714, 2020.

[26] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 619–631, New York, NY, USA, October 2017. Association for Computing Machinery.

[27] Donghang Lu, Albert Yu, Aniket Kate, and Hemanta Maji. Polymath: Low-Latency MPC via Secure Polynomial Evaluations and Its Applications. *Proceedings on Privacy Enhancing Technologies*, 2022(1):396–416, January 2022.

[28] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its Application to Anonymous Communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 887–903, New York, NY, USA, November 2019. Association for Computing Machinery.

[29] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522, 2020.

[30] 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, May 2017.

[31] Lucien K. L. Ng and Sherman S. M. Chow. {GForce}: {GPU-Friendly} Oblivious and Rapid Neural Network Inference. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2147–2164, 2021.

[32] Lucien K. L. Ng and Sherman S. M. Chow. SoK: Cryptographic Neural-Network Computation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 497–514, May 2023.

[33] Deevashwer Rathee, Thomas Schneider, and K. K. Shukla. Improved multiplication triple generation over rings via rlwe-based ahe. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *Cryptology and Network Security*, pages 347–359, Cham, 2019. Springer International Publishing.

[34] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015.

[35] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1021–1038, May 2021.

[36] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, volume 16, pages 601–618, 2016.

A Evaluation of BatchNorm with Polynomials

	Without BatchNorm	With BatchNorm
Normal Relu	88.77 ± 0.11	90.05 ± 0.16
PolyRelu	82.99 ± 0.42	87.37 ± 0.13

Table 7: Comparing the effect of BatchNorm on MiniONN model.

To study the effect of a batch norm layer on our training process, we train a standard ReLU model and a model with polynomial activations both with and without batch norm layers. We use the MiniONN architecture and give the results averaged over three random seeds with 95% confidence intervals in Table 7. We find that batch norm layers improve both models. However, the improvement due to using batch norm is significantly greater when using polynomial activation functions. This is an intuitive result as batch norm helps keep each layer's output bounded and thus reduces the work of our regularization function.

B Evaluation of Sigmoid with Polynomials

	ReLU	Sigmoid
Standard Activation	94.7 ± 0.08	90.2 ± 0.11
Polynomial Approx.	93.4 ± 0.16	85.9 ± 0.11

Table 8: Comparing the effect of the activation function on a ResNet18 model.

In this work, we focus on the ReLU activation function, the default in common architectures such as ResNets [16]. Another reason we focus on ReLU is that it gives better accuracy than alternatives like Sigmoid. In this section, we highlight this accuracy advantage by the accuracy of a ResNet18 model with different activation functions. In Table 8, we evaluate

both ReLU and Sigmoid with and without using polynomial approximation. In all cases, we use the ResNet18 architecture with default parameters given in Section 6.1, we note the polynomial is of degree $n = 4$. The results are averaged over five random seeds and shown with 95% confidence intervals. We find that ReLU consistently outperforms Sigmoid with and without using polynomial evaluations. However, the accuracy of Sigmoid decreases more, relative to ReLU, when using polynomial approximation. This result further motivates our use of ReLU. We leave further investigation of Sigmoid and other activations for future work.

C Quantization Aware Polynomial Fitting

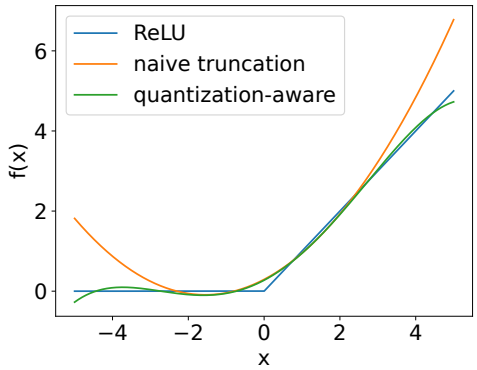


Figure 11: The effect of truncation on a polynomial activation function.

In Figure 11, we plot the polynomial approximation with and without our quantize-aware fitting approach described in Section 5.1. First, we plot the polynomial approximation after truncation and see that it diverges from a true ReLU. We also plot our quantized polynomial fitting and show that it addresses the problems of exploding activations within the range.

Polynomial Input Distribution After Regularization We plot the histogram of the input to all activation functions of a ResNet 18 model on CIFAR-10 in Figure 12.

D Communication and Rounds Benchmark

We study two additional evaluation metrics of rounds and communication in this section. Recall that the number of rounds significantly affects the protocol latency over WAN. The communication affects each round's throughput, depending on the network bandwidth. In Table 9, we summarize the communication in GB and the number of rounds (shown in parenthesis) for our work and the related work we compare to in Section 6. We note that both COINN and GForce do not

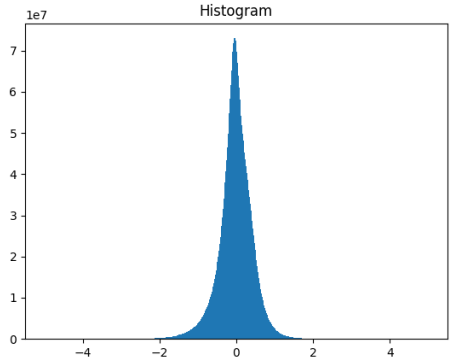


Figure 12: Polynomial Input Distribution of ResNet18 on CIFAR10.

log the communication or rounds in their code base. Thus, we report their communication numbers from the corresponding tables in the papers (COINN [18, Table 3] and GForce [31, Table 7]). Neither work evaluates the number of rounds.

In all cases, ESPN and HoneyBadger significantly dominate all evaluated related work in the number of rounds with a $3-5\times$ improvement. The communication of ESPN and HoneyBadger is significantly less than COINN, approximately the same as CryptGPU, and more than GForce and Cheetah. Our HoneyBadger solution has approximately $2\times$ the communication of Cheetah, but Cheetah has $10\times$ the rounds; thus, in practice, HoneyBadger gives much better performance as shown in Section 6. GForce has an impressively low communication online (50MB) due to offloading 20GB of communication to an offline phase. Once again, despite having higher communication, we recall that both ESPN and HoneyBadger dominate related work in runtime in the WAN (as shown in Section 6).

E Correctness Proofs

We begin by proving the correctness of Algorithm 1.

Theorem E.1. *Given an input $[[x]] = x_A + x_B$ and exponent k, Algorithm 1, correctly returns $[[x^k]]$.*

Proof. We begin with party A. In line 4, they compute their share of the vector $\mathbf{a}$ where $\mathbf{a}_i = x_A^{k-i}$ (since $\mathbf{a}$ was initialized to zero and the for loop iterates over each entry of $\mathbf{a}$ exactly once). Similarly, in line 5, party B computes their share of $\mathbf{b}$ where $\mathbf{b}_i = \binom{k}{i} x_B^i$. We recall that party B's share of $\mathbf{a}$ is the zero vector, and similarly for party A's share of $\mathbf{b}$ Then, the vector $\mathbf{p}$ is obtained by multiplying $\mathbf{a}$ and $\mathbf{b}$ element wise in line 6. Thus, $\mathbf{p}_i = \mathbf{a}_i \cdot \mathbf{b}_i = x_A^{k-i}\binom{k}{i}x_B^i$. The final step (line 7), simply sums $\mathbf{p}$. Therefore,

$$s = \sum_{i=1}^{k} \mathbf{p_i} = \sum_{i=1}^{k} \binom{k}{i} x_A^{k-i} x_B^i \tag{8}$$

Dataset	Model	ESPN	HoneyBadger	CryptGPU	COINN	GForce	Cheetah
CIFAR-10	ResNet-18	0.46 (38)	0.23 (38)	0.45 (174)	/	/	/
	ResNet-110	0.55 (221)	0.13 (221)	0.53 (1093)	6.8	/	/
	VGG-16	0.32 (31)	0.26 (31)	/	/	0.050	/
CIFAR-100	ResNet-18	0.46 (38)	0.23 (38)	0.45 (174)	/	/	/
	ResNet-32	0.16 (65)	0.037 (65)	0.15 (313)	1.9	/	/
	VGG-16	0.32 (31)	0.26 (31)	/	/	0.050	/
ImageNet	ResNet-50	7.85 (160)	4.04 (160)	7.70 (552)	122.0	/	2.36 (1042)

Table 9: Evaluation of communication in GB and the number of rounds (shown in parenthesis).

which applying the binomial theorem (2) gives $(x_A + x_B)^k = [[x^k]]$. □

Given the correctness of Algorithm 1, we now prove the correctness of Algorithm 2. To begin, we bound the failure probability of each truncation step in Algorithm 2.

Theorem E.2. *Consider computing a degree n polynomial fitted to the range $[-\lambda, \lambda)$. Let the global precision (size of the ring) be L-bit and the working precision of each value be p-bit. Then, the truncation in line 5 of Algorithm 2, fails (The local division of $[[x]_A, [x]_B] * 2^{-\bar{s}} \neq [[x * 2^{-\bar{s}}]]$) with probability at most*

$$Pr[\text{Line 5 Failure}] \leq \frac{2^{\lceil \log_2 \lambda \rceil + 1 + p}}{2^L}. \tag{9}$$

Proof. Consider the first truncation of Algorithm 2 in line 5. The input to this truncation is x which we assume is contained in the range $[-\lambda, \lambda)$ with a working precision of p-bits. Therefore, the size of x is $2^{\lceil \log_2 \lambda \rceil + 1}$ in the integer part and 2^p in the decimal part. Which gives, $|x| \leq 2^{\lceil \log_2 \lambda \rceil + 1 + p}$. Given that we are working in a L-bit ring and the probability of failure of the truncation protocol is bounded by $|x|/Q$ where Q is the ring size [22], the result follows. □

Theorem E.3. *Consider computing a degree n polynomial fitted to the range $[-\lambda, \lambda)$. Let the global precision (size of the ring) be L-bit and the working precision of each value be p-bit. Then, the truncation in line 7 of Algorithm 2 fails (The local division of $[[x]_A, [x]_B] * 2^{-p*(i-1)+\bar{s}\cdot i} \neq [[x * 2^{-p*(i-1)+\bar{s}\cdot i}]]$) with probability at most*

$$Pr[\text{Line 7 Failure}] \leq \frac{2^{i(\lceil \log_2 \lambda \rceil + 1) + 2p}}{2^L}. \tag{10}$$

where i is the power of x being truncated ($i \leq n$).

Proof. Consider the truncation in line 7 of Algorithm 2. The input to this truncation is the output of the previous truncation in line 5, raised to the power i. We assume the previous truncation was correct. Then after the truncation,

$$|x'_i| = \frac{2^{\lceil \log_2 \lambda \rceil + 1 + p}}{2^{\lceil (i-2)p/i \rceil}} \leq \frac{2^{\lceil \log_2 \lambda \rceil + 1 + p}}{2^{(i-2)p/i}} \tag{11}$$

for $i \in \{2, \ldots, n\}$, where the inequality holds because $(i-2)p/i$ is positive. After applying the exponentiation by i we get $|\mathbf{p}_i| \leq 2^{i(\lceil \log_2 \lambda \rceil + 1) + 2p}$. Given that we are working in a L-bit ring and the probability of failure of the truncation protocol is bounded by $|x|/Q$ where Q is the ring size [22], the result follows. □

Theorem E.4. *Given an input $[[x]] = x_A + x_B$ and polynomial coefficients α, Algorithm 2, correctly returns the polynomial evaluation $[[\sum_{i=0}^{n} \alpha_i \cdot x^i]]$ except with probability.*

$$Pr[\text{Alogrithm 2 Fails}] \leq \frac{\sum_{i=2}^{n} \left(2^{i(\lceil \log_2 \lambda \rceil + 1) + 2p} + 2^{\lceil \log_2 \lambda \rceil + 1 + p}\right)}{2^L}.$$

Proof. First, prove correctness assuming the truncation operators are correct (then we will account for the probability of failure). We begin by proving that $\mathbf{p'_i} = x^i$ for $i \in \{1, \ldots, n\}$ after line 7. We note that x is actually $x * 2^p$ due to the fixed point encoding. $\mathbf{p_1}$ follows trivially. For $i \in \{2, \ldots, n\}$, we work backwards, from line 7 to line 4 expanding the definition of $\mathbf{p'_i}$

$$\begin{aligned}
\mathbf{p'_i} &= \mathbf{p}_i * 2^{-p*(i-1)+\bar{s}\cdot i} && (12)\\
&= (x'_i)^i * 2^{-p*(i-1)+\bar{s}\cdot i} && (13)\\
&= (x \cdot 2^p \cdot 2^{-\bar{s}})^i * 2^{-p*(i-1)+\bar{s}\cdot i} && (14)\\
&= x^i \cdot 2^p && (15)
\end{aligned}$$

where the first lines, up to (14), come from the substitution of lines 7 through 5, respectively. The remaining step follows from basic algebra.

Finally, we bound the failure probability of the algorithm. The truncation steps in line 5 and line 7 each introduce a possibility of wrap around error. Each truncation is executed $n-1$ times. Thus, applying the bounds derived in Theorem E.2 and Theorem E.3 for each i in the for loop, the total failure probability follows. □