Approximating Complex Neural Network Activation Functions for Real Time Secure Computation

Mazharul Islam*
University of Wisconsin-Madison

Sunpreet S. Arora
Visa Research

Rahul Chatterjee
University of Wisconsin-Madison

Peter Rindal
Visa Research

Maliheh Shirvanian*
Netflix

ABSTRACT

There are primarily two different ways in which deep learning models are being deployed in practical applications: on the edge (client) device directly or as a software service in the cloud. Deployments on the edge device, if not secured properly, run the risk of model inversion or the client learning sensitive information from the model. Cloud deployments, on the other hand, often require the client to share their private data with the cloud service for the inference task. Privacy-preserving techniques, such as secure multi-party computation (MPC), can be used to address the aforementioned privacy risks. While state-of-the-art MPC techniques are compatible with popular activation functions (e.g., ReLU), they do not generalize efficiently to more complex activation functions (e.g., SiLU) that are routinely used in state-of-the-art deep learning models. This is because such complex activation functions are highly non-linear in nature. We propose a scheme that approximates complex activation functions using a set of piece-wise polynomials for privacy-preserving inference. The approximations are designed to be compatible with efficient MPC techniques useful for privacy-preserving inference, resulting in an approach that does not require retraining of deep learning models. We conduct extensive evaluations by integrating our approach with state-of-the-art deep learning models designed for various machine learning tasks. Our experimental results show that our scheme generated MPC-friendly activation functions do not significantly degrade inference accuracy of the underlying model nor increase latency of the MPC scheme.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Deep neural networks, Complex activation functions, Secure evaluation

*Work done while at Visa Research

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Mazharul Islam, Sunpreet S. Arora, Rahul Chatterjee, Peter Rindal, and Maliheh Shirvanian. 2022. Approximating Complex Neural Network Activation Functions for Real Time Secure Computation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep neural network (DNN) based models are being increasingly adopted in a variety of practical applications. Examples include autonomous driving systems (e.g., Tesla's Autopilot[1] and General Motor's Super Cruise[2]) that use image classification and object detection models and digital assistants (e.g., Google Assistant[3] and Amazon Alexa[4]) that leverage speech translation models.

Deep learning models used in real-world applications are deployed either directly on the client/edge device or in a cloud environment. Models provisioned on the edge are advantageous from the following perspectives: they do not require the edge device to share the inference data directly with a server hosted in a cloud environment, and the latency is minimal because communication between edge device and cloud based server is not required at the time of inference. However, model deployments at the edge may not be favorable for deep learning service providers because edge devices have limited compute and storage capabilities, and may limit the ability of service providers to update the models frequently. In addition, proprietary models if not secured properly may leak sensitive information potentially resulting in business consequences for the deep learning service providers. For example, a malicious client could execute model extraction attacks [26] or model inversion attacks [13] to learn proprietary training data or model architecture.

Cloud based deployments address the aforementioned limitations of edge deployments and allow compute and storage scalability to serve hundreds of thousands of queries. However, client-server communication at the time of inference introduces additional latency. Additionally, such services require the client to share their data with the service provider for the inference task. This has privacy implications if client's data is private or sensitive (e.g., healthcare records or biometric data). For this scenario, a fundamental question to address is: given a proprietary deep learning model trained on sensitive or private data, *is it possible to make predictions obliviously* where clients gets the prediction result without revealing anything about their private data to cloud service and

[1]https://www.tesla.com/en_eu/support/autopilot
[2]https://www.gmc.com/connectivity-technology/super-cruise
[3]https://assistant.google.com/
[4]https://alexa.amazon.com/

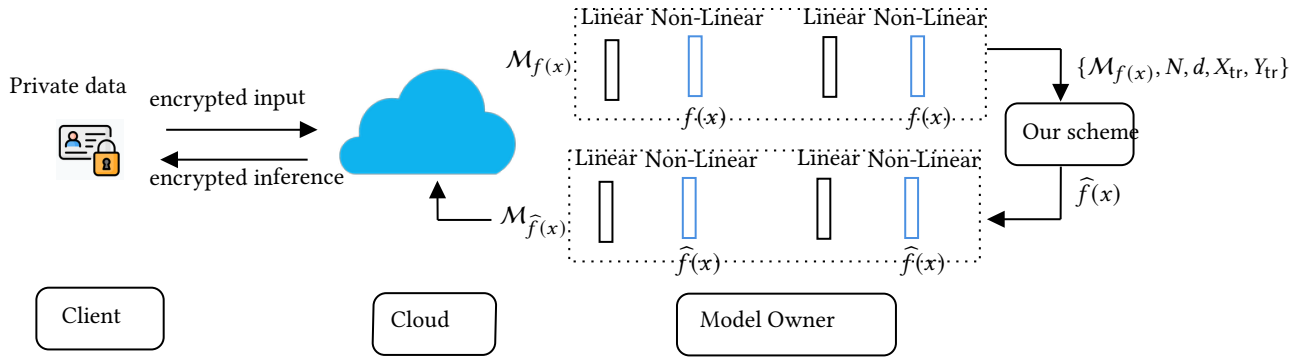

Figure 1: Secure inference problem in MLaaS. In this setting, we denote the already trained DNN model $\mathcal{M}$ by the model owner where non-linear layers comprise of complex activation function $f(x)$ as $\mathcal{M}_f$. Model owner will replace $f(x)$ with its MPC-friendly version $\widehat{f}(x)$ and denote it by $\mathcal{M}_{\widehat{f}}$ (before deployment). Consequently $\mathcal{M}_{\widehat{f}}$ becomes MPC-friendly since all operations in linear layers and non-linear layers can be done via any state-of-the-art MPC library. After deployment model owner can run secure inference over client can encrypt their private data and send the encrypted inference result to client which it can decrypt.

service providers maintaining the secrecy of their propriety DNN models.

Recent progress in secure two party computation (2PC) and secure multi-party computation (MPC) has inspired a number of work to solve this problem — commonly referred to as "secure inference" [1, 8, 9, 14, 15, 24, 28, 33, 41**?** , 42]. In this line of work, clients encrypt their input before uploading their data and service providers have to do secure inference over clients' encrypted inputs. However, leveraging existing clever MPC cryptographic techniques to solve this problem faces a fundamental challenge; 2PC and MPC techniques such as oblivious transfer (OT), homomorphic encryption (HE), secret-sharing are especially optimized for computing linear functions efficiently. But for evaluating non-linear functions it falls short. This is problem since each layer of the DNN models comprise of both linear as well as non-linear components (Figure 3). As a result these non-linear components of each layers presents a challenge to computer securely using existing techniques.

Prior works have proposed a number of solutions attempting to optimizate for both linear and non-linear components of DNN models for secure inference. Unfortunately, they suffer from at least one of the following major limitations: (1) they require making changes to how the models are trained [9, 15, 36, 42]; (2) requires the presence of a designer who has to configure certain operations for accuracy vs performance trade-offs [14, 33]; (3) specifically tailored towards DNN models which only relies on ReLU activation functions [8, 28, 41]. (4) they are experimented on shallow neural networks and when experimented on deep DNN their prediction accuracy degrades significantly [30, 35, 36, 43];

In this paper, we set out to answer these limitations, by first developing novel schemes to securely evaluate more complex non-linear activation functions — that are routinely encapsulated in state-of-the-art DNNs (e.g., GPT-3, BERT, Efficient-Net) for better performance and generalization. Our techniques can take a complex activation function and generate a piece-wise polynomial the approximation that is easy to compute using any MPC techniques that supports $+, \times, >$ operations. Recently Fan et al. (CCS '22) [11] proposed NFGen, an independent and concurrent work of ours to generate MPC friendly version given non-linear functions by approximating it using a number of piece-wise polynomials that can be computed efficiently using MPC platforms that support $+, \times, >$ operations. Our design, although sharing similar philosophy, differs from their proposed techniques in a number of ways. First, our work instead of approximating the original activation functions our scheme introduces crude approximations, that have a high approximation error. However, this error introduced by the crude approximation is much easier to approximate using piece-wise polynomials than the original non-linear activation functions (Section 4.1). Secondly, our approximation is input density aware where we can calculate a prior over the input to the activation functions. This is possible for DNN scenarios before forwarding input to the activation functions, normalization is introduced that puts the input within a certain range (Section 4.2). Finally, we realize instead of placing each piece-wise polynomial equally spaced an approximation error can be reduced largely by having more closely spaced polynomials where there is a high error and distance spaces polynomials where there is less error (Section 4.3). Our DNN model's specific realizations resulted in designing MPC-friendly piecewise polynomial recipes for three popular complex activation functions sigmoid, SiLU, GeLU. We show via extensive evaluation that our generated MPC-friendly approximations when used for secure inference have a negligible loss in accuracy.

To demonstrate the applicability of proposed techniques We incorporated our scheme with a popular secure two-party computation library MP-SPDZ [25] and secret sharing-based MPC platform ABY^3 [35] to benchmark the efficiency of secure inference for two and three-party scenario respectively as shown in Figure 1 Our results show that for 2PC scenario it adds a maximum delay of $< 1,023$ ms and for MPC scenario with three parties < 500 ms (Section 5.2.1).

Our contributions. In summary, we make the following contributions in the paper.

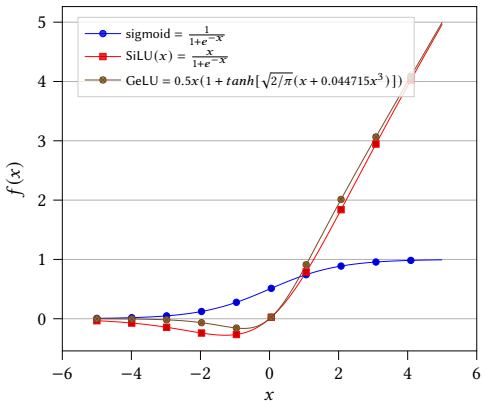

Figure 2: Complex activation functions sigmoid, SiLU, GeLU **for input** $x \in [-5, +5]$**.**

- We design MPC-friendly piece-wise polynomial approximation for highly complex non-linear activation functions routinely used in wide and deep neural networks with negligible loss in inference accuracy. Our DNN-specific optimization resulted in a MPC-friendly approximation that does not degrade inference accuracy.
- By incorporating our design with two popular MPC platform, we show the feasibility of secure inference for applications that requires the inference results in real-time for both two-party and three-party scenario.
- We implement our designed MPC-friendly activation functions[5] and show the efficiency of our design in terms of performance accuracy and prediction time across different types of tasks, DNN models and datasets.

2 BACKGROUND AND RELATED WORK

2.1 Deep Neural Network Preliminaries

2.1.1 Complex Activation Functions. Activation functions are one of the most crucial components for DNNs used for adding non-linearity to the learning process. Researchers have shown an activation function with non-monotonicity nature, ability to handle small negative weights and smooth curve can increase the learning convergence of the model while training with back-propagation, as well as learning a nonlinear classifier that can do predictions with high accuracy. Early machine learning models used to relay on simple binary threshold units [12, 20] and latter switched to sigmoid and TanH for their smoothness [**?**]. Despite having less statistical motivation ReLU activation function that makes hard gating decision based on inputs sign showed remarkable faster convergence and prediction accuracy [27, 37]. However, ReLU is not without its pitfalls, as researchers have tried to address the limitations of ReLU by introducing LeakyReLU [32], DyingReLU [31], and parametric ReLU [48].

Recent work based both theoretical and empirical understanding has paved the way to new advanced complex activation functions

[5]Pytorch implementation is available at https://github.com/islamazhar/Cilantro-Public

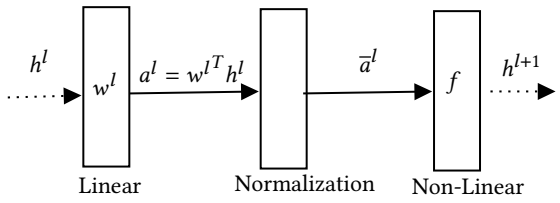

Figure 3: Linear, normalization, and non-linear operation are performed sequentially for the l^{th} **layer. Results of linear operation** a^l **are normalized to** $\overline{a}^l$ **either batch wise or layerwise before applying the non-linear activation function** f**.**

in the areas of computer vision, natural language processing, and speech tasks. These new activation functions such as SiLU [10], GeLU [18], Mish [34], ELU [] exhibiting superior performance, are more smooth, can handle small weights and as a result perform better than ReLU in terms of fast convergence and prediction accuracy. Unfortunately, unlike ReLU which is easy to compute for secure evaluation, these new functions are complex, more non-linear as shown in Figure 2 and hard to evaluate using MPC techniques. In this work address this limitation by designing new MPC-friendly recipe for these complex activation functions.

Lastly, a few interesting works although orthogonal to ours, have looked at improving the learning capability of Lipschitz constraint neural networks by replacing ReLU with a learnable activation comprising of a set of linear splines that introduces more expressiveness [3, 5, 6, 38].

2.1.2 Normalization. To address the issue of "internal covariance shift" where a layer's input distribution changes abruptly due to its dependency on previous layers, normalization was proposed. Normalization also brings stability to the training process dissolves many problems while training DNN models (e.g., dependence on initial parameters selection, requiring lower running rates, etc.). Normalization is performed before the non-linear components of a layer when the outputs of the linear layers are forwarded as inputs to the non-linear layer. Figure 3 illustrates the normalization of l^{th} layer having H dimension and B batchsize for inputs $h^l \in [H \times B]$. Inputs to the normalization component are results of the linear component $a^l = w^{lT} h^l$. Then a^l are normalized to $\overline{a}^l$ according to a calculated mean μ and std. deviation σ before applying the non-linear function f. There are two popular normalization techniques used in state-of-the-art DNN models as discussed briefly following.

- Batch Normalization (BN): BN was first proposed by Ioffe and Szegedy in their seminal work [22] where they performed normalization batchwise (i.e., row-wise for matrix a^l). Informally, we first calculate mean $\boldsymbol{\mu}_{\text{BN}}$ and variance $\boldsymbol{\sigma}_{\text{BN}}$ over all inputs present in the a batch of size B. To normalizate the calculated $\{\boldsymbol{\mu}_{\text{BN}}, \boldsymbol{\sigma}_{\text{BN}}\}$ are used to normalize each dimension of the l layer as shown in Eqn 1.
- Layer Normalization (LN): Closely following BN, Ba et al. proposed LN [4] where normalization is performed layerwise (i.e., colum-wise for matrix a^l). LN makes it possible to apply normalization even when batch size is 1 which is important models which are trained sequence by sequence (such as

RNN). For the case of LN, $\{\boldsymbol{\mu}_{\mathsf{LN}}, \boldsymbol{\sigma}_{\mathsf{LN}}\}$ are calculated over single input present in the same layer as shown in Eqn 2.

Since normalization is applied for state-of-the-art DNN models, we can leverage this technique to have a good estimation of the input density to the complex activation functions in DNN models.

$$\boldsymbol{\mu}_{\mathsf{BN}}, \boldsymbol{\sigma}_{\mathsf{BN}} = \bigcup_{i=1}^{H} \{\mu_i, \sigma_i\} \Big| \mu_i = \frac{1}{B}\sum_{b=1}^{B} a^l_{i,b} \text{ and } \sigma_i = \frac{1}{B}\sqrt{\sum_{b=1}^{B}(a^l_{i,b} - \mu_i)^2} \tag{1}$$

$$\boldsymbol{\mu}_{\mathsf{LN}}, \boldsymbol{\sigma}_{\mathsf{LN}} = \bigcup_{b=1}^{B} \{\mu_b, \sigma_b\} \Big| \mu_b = \frac{1}{H}\sum_{i=1}^{H} a^l_{i,b} \text{ and } \sigma_b = \frac{1}{H}\sqrt{\sum_{i=1}^{H}(a^l_{i,b} - \mu_b)^2} \tag{2}$$

2.2 Secure Inference for DNN models

Recent advances in MPC technique to compute over encrypted data give a compelling solution to secure inference problem. Client can encrypt their input, and send it to cloud services which can run inference by the trained DNN models over the encrypted input. However, MPC techniques have been optimized for linear operations. Therefore computing non-linear operations involved during the secure inference turns out to be a fundamental challenge. In the following few sections, we highlight how prior works have attempted to solve this problem.

2.2.1 Secure inference for ReLU *specific DNN.* During prediction, DNN models need to apply both linear and non-linear transformations of the input in an alternating way. Existing MPC frameworks can handle linear transformations of the input such as addition, and multiplication efficiently. However, they fall short when computing non-linear transformation securely, in particular non-linear activation function.

To address this problem prior works generally focus on ReLU activation function which is easy to handle in MPC [8, 21, 41, 46, 47]. For example, Rathee et al. proposed new 2PC protocols for secure comparison and division to evaluate ReLU efficiently by reducing the communication cost significantly [41] in semi-honest settings. Follow-up work improved the threat model to malicious client setting [8, 21]. However such optimizations specialized for ReLU do not work for other complex activation functions.

Another line of work has relied on Garbled Circuits (GC) to evaluate the non-linear part securely [23, 28, 33, 43]. However, all of the have been experimented on at most shallow DNN models (less than 7 layers) having ReLU activation functions. Therefore it remains to be seen if these techniques can do real-time inference for wide DNN models having complex activation functions as mentioned in Section **??**.

Another technique to handle the non-linear activation function efficiently is to apply restrictions the way the network is trained. For example Riazi [42] et al. leverage GC-based protocol to do secure inference for binary neural networks (BNN). However, as mentioned in Section **??**, retraining the propriety models from scratch and spending more money is an unfeasible solution and discouraging model owners from deploying secure inference services.

Lastly Pereteanu et al. [39] proposed revealing middle part of the DNNs with the idea that client can not infer the last and first part of the DNNs but will improve the communication complexity. However it is unlikely that model owners are willing to reveal any part to the propriety models clients especially when both model and training datasets are privacy sensitive and any leak can cause huge loss revenue loss.

In particular, a plethora of research work has improved the state-of-the secure inference for ReLU based shallow DNNs. Our work is a crucial next step toward focusing on other complex activation functions which have outperformed ReLU and now getting traction in ML community.

2.2.2 Secure inference for more complex non-linear activation functions. A common approach to handle non-linear activation functions is to approximate them with piece-wise polynomial functions. These piece-wise polynomial functions are easy to compute for MPC frameworks and thus are MPC-friendly. The challenge here is not to degrade the prediction accuracy as approximation error can cause incorrect prediction. For example, Delphi [33] runs a planner that balances which activation function can be replaced with low-degree polynomials without introducing too many inaccuracies and gain a significant communication benefit. In similar sense CryptoNet [15], SecureML [36], CryptoDL [19], ABY^3 [35], Minion [30] also, pursue similar ideas. However, they are application specific, and switching to another application degrades accuracy significant [14] due to small errors getting propagated resulting in NaNs value. In addition, they are heavily focusing on ReLU activation functions and therefore as we will show in Section **??** using their recipe of piece-wise polynomial approximation performs poorly for other complex activation functions incurring a significant loss in accuracy — sometimes worse than a random classifier.

3 PROBLEM FORMULATION

In secure inference problem, we assume the model owner has already trained a DNN model $\mathcal{M}_f$ on dataset $X_{\text{tr}}, Y_{\text{tr}}$. Each layer of $\mathcal{M}$ comprises of three sequential components linear, normalization, and non-linear having complex activation function f as shown on Figure 3. Since existing MPC libraries are unable to handle non-linear components efficiently, to enable secure inference over client encrypted input, model owner will replace f with its MPC-friendly version $\widehat{f}$. Note the other two components linear, and normalization can already be performed using existing MPC techniques efficiently. We pictorially present this problem in Figure 1.

Ideally, $\widehat{f}$ should have the following two characteristics 1) it should not degrade the prediction quality and 2) It should affect the inference latency during secure inference compared to plaintext inference without encrypting client's input. To generate such ideal $\widehat{f}$, we assume model owner will choose two parameters 1) N the number piece-wise polynomials considered for $\widehat{f}$ and 2) d maximum degree of each of the N polynomials. Therefore for a given $\left\{\mathcal{M}_f, X_{\text{tr}}, Y_{\text{tr}} N, d\right\}$, for secure inference, we want to find a MPC-friendly piece-wise polynomial of $\widehat{f}(x)$ of $f(x)$ such that $\widehat{f}$ consists of N piece-wise polynomials and the degree of each of these polynomials is at most of degree d. We want to find the $\widehat{f}(x)$ among all possible piece-wise approximations $\mathcal{F}$ that would minimize the

Symbol	Description of the symbol
$f(x)$	Complex activation functions.
$\widehat{f}_{\text{crude}}$	Crude MPC friendly approximation of $f(x)$.
$\widehat{f}_{\text{improved}}$	Improved MPC friendly approximation of $f(x)$.
$[x_1, x_2, \cdots, x_n]$	set of n points considered for MPC-friendly approximation of $f(x)$.
N	# of piece-wise polynomials used for approximation.
d	Maximum degree of each of N pieces.
$\text{poly}^d_{[i,j]}(x)$	Polynomial to interpolate the approximation error between $[i, j]$ using degrees $\leq d$.
$P(x)$	Probability distribution of x.
δ_{crude}	Approximation error introduced by $\widehat{f}_{\text{crude}}$.
δ_{thr}	Maximum threshold for approximation error

Figure 4: Common notations used in this work.

Input: $P(x), d, N, \widehat{f}_{\text{crude}}, [x_1, x_2, \cdots, x_n]$
Output: $\widehat{f}_{\text{improved}}(x)$

$\delta_{\text{thr}} \leftarrow \frac{1}{N} \cdot \texttt{ApproxError}\left(f, \widehat{f}_{\text{crude}}, (x_1, \ldots, x_N), P\right)$
$\widehat{f}_{\text{improved}} \leftarrow \phi$
$x_s \leftarrow x_1$
forall $i = 2 : n$ **do**
  $\delta_{\text{crude}} \leftarrow \texttt{ApproxError}\left(f, \widehat{f}_{\text{crude}}, [x_s, x_{s+1}, \cdots, x_i], P(x)\right)$
  if $\delta_{\text{crude}} > \delta_{\text{thr}}$ **then**
    $\text{poly}^d_{(x_s,x_i)} \leftarrow \texttt{Interpolate}\left(f, \widehat{f}_{\text{crude}}, d, [x_s, x_i]\right)$
    $\widehat{f}_{\text{improved}} \leftarrow \widehat{f}_{\text{improved}} \cup \text{poly}^d_{(x_s,x_i)}$
    $x_s \leftarrow x_i$
return $\widehat{f}_{improved} \cup \widehat{f}_{crude}$

Algorithm 1: Protocol to design secure efficient MPC friendly version of an activation function $f(x)$. Notations used are described in Figure 4.

Input: $f(x), \widehat{f}_{\text{crude}}(x), [x_i, x_{i+1}, \cdots, x_j], P(x)$
Output: δ */*density aware approximation error*/*

$\delta \leftarrow \sum_{x_i}^{x_j} P(x) \cdot \left| f(x) - \widehat{f}_{\text{crude}}(x) \right|$
return δ

Algorithm 2: Function to calculate density aware approximate error.

absolute difference between evaluating the prediction quality on evaluating the client input X_{ts}, on $\mathcal{M}_f$ (plaintext inference) and $\mathcal{M}_{\widehat{f}}$ (secure inference) as shown on Eqn. 3.

$$\min_{\widehat{f} \in \mathcal{F}} \left| \texttt{Eval}(\mathcal{M}_{\widehat{f}}, X_{\text{ts}}, Y_{\text{ts}}) - \texttt{Eval}(\mathcal{M}_f, X_{\text{ts}}, Y_{\text{ts}}) \right| \quad \text{s.t. } \widehat{f} = \bigcup_{i=1}^{N} \text{poly}_i \text{ and } \forall i \ \deg(\text{poly}_i) \leq d \tag{3}$$

Input: $(x_i, x_j), f(x), \widehat{f}_{\text{crude}}(x), d$
Output: $\text{poly}^d_{(x_i,x_j)}$

if interpolation type = *"Chebyshev"* **then**
  $\text{poly}^d_{(x_i,x_j)} \leftarrow \texttt{Chebyshev}\left((x_i, x_j), f, \widehat{f}_{\text{crude}}, d\right)$
else // Interpolation type is BSpline
  $Y \leftarrow P(x_i) \cdot \left(f(x_i) - \widehat{f}_{\text{crude}}(x_i)\right)\Big|$ for all $x_i \in X$
  $\text{poly}^d_{(x_i,x_j)} \leftarrow \texttt{BSpline}(X, Y, d)$
return $\text{poly}^d_{(x_i,x_j)}$

Algorithm 3: Interpolating a polynomial to approximate the crude approximation error

4 OUR PROTOCOL OVERVIEW

To enable efficient secure inference of client input x from DNN model relying on complex activation functions f, held by the model owner, we need to develop protocols for MPC-friendly version of f. To this end we take inspiration from [30, 35, 36] which suggested generating MPC-friendly version of one particular complex activation functions sigmoid by approximating it using piece-wise linear polynomials. Throughout the paper we use a hat sign $\widehat{(.)}$ over $f(x)$ to represent its MPC-friendly version of an activation function $\widehat{f}(x)$. A summary of the notations frequently used in this paper is shown in Figure 4.

Formally Our scheme as shown on Figure 1, takes as input an complex activation function $f(x)$ and a prior over the probability density of input x denoted by $P(x)$. We also assume we have handcrafted a piece-wise MPC friendly crude but simple approximation of f with the property that it may have high crude approximation error within domain $[x_s, x_e]$ but outside this domain when $x < x_s$ or $x > e$ it is zero. We denote such crude MPC-friendly approximation by $\widehat{f}_{\text{crude}}$ and crude approximation error by $\texttt{ApproxError}(x) = f(x) - \widehat{f}_{\text{crude}}(x)$. We demonstrate how efficiently $\widehat{f}_{\text{crude}}$ and $P(x)$ can be constructed in Section 4.1 and Section 4.2 respectively in details. Apart from these three inputs, our scheme also takes two other additional inputs that dominate the inference time that is 1) the number of pieces to consider for approximating $\texttt{ApproxError}(x)$ (denoted by N) and 2) the maximum degree to consider during polynomial interpolation (denoted by d). We show how the values of N and d can be derived empirically in Section **??**.

Given the above-mentioned five inputs as shown in Figure 1, our scheme will output a MPC friendly version $\widehat{f}_{\text{improved}}(x)$ for complex activation function $f(x)$ such that when f is replaced by MPC friendly version $\widehat{f}_{\text{improved}}(x)$ in DNN during for secure inference, prediction accuracy does not decrease and does not add too much delay. Specifically $\widehat{f}_{\text{improved}}(x)$ is MPC friendly since it comprises of $\widehat{f}_{\text{crude}}$ — the initial MPC friendly crude approximation of $f(x)$ (given as input) and a list of N piece-wise polynomials $\bigcup_i \text{poly}^d_{(x_s,x_i)}$ of degree at most degree d covering the input domain from x_s to x_e and $\texttt{ApproxError}(x)$. A few remarks to follow here.

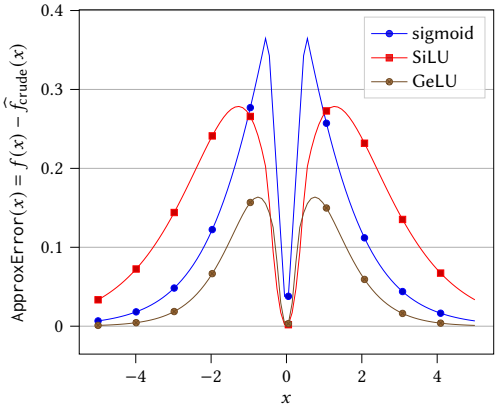

Figure 5: $\texttt{ApproxError}(x)$ for three complex non-linear activation functions sigmoid, SiLU, GeLU **where $\widehat{f}_{\text{crude}}$ for these are defined in Eqn. 4, 5, 6.**

Firstly the reason to use these N piece-wise polynomials to approximate $\texttt{ApproxError}(x)$ introduced by $\widehat{f}_{\text{crude}}$; instead of trying to use them to approximate the original f which seems to be a more natural choice adopted by prior work [11, 16, 30, 36] is, as we will describe, for certain choices of $\widehat{f}_{\text{crude}}$ it is much easier to approximate δ_{crude} as opposed to approximate the original complex and highly non-linear $f(x)$. Secondly, we say $\widehat{f}_{\text{improved}}(x)$ is an improvement over $\widehat{f}_{\text{crude}}$ because it compensates the crude approximation error $\texttt{ApproxError}(x)$ via the list of N piece-wise polynomials. Lastly, similar to $\widehat{f}_{\text{crude}}$, $\widehat{f}_{\text{improved}}(x)$ is also MPC friendly since it can be computed via any MPC library that supports $+, \times, >$ operations.

4.1 Designing $\widehat{f}_{\text{crude}}$

One of the first steps to designing MPC-friendly piece-wise polynomials for a complex activation function $f(x)$ using our scheme is to come up with crude approximation $\widehat{f}_{\text{crude}}(x)$ that is 1) MPC friendly and 2) may have high approximation error $\texttt{ApproxError}(x) = f(x) - \widehat{f}_{\text{crude}}(x)$ but $\texttt{ApproxError}(x)$ is easy to approximate instead of approximating the original $f(x)$ using piece-wise polynomials. To this extent, for sigmoid, we adopted, a similar but slightly changed for getting better performance, piece-wise linear approximation proposed in [36] and call it $\widehat{f}^{\text{sigmoid}}_{\text{crude}}$ (Eqn. 4).

$$\widehat{f}^{\text{sigmoid}}_{\text{crude}} = \max\left(0, \min(\frac{x}{3}, \frac{1}{2})\right) \tag{4}$$

For the other two complex activation functions SiLU and GeLU, we design two more crude approximations (Eqn. 5,6)

$$\widehat{f}^{\text{SiLU}}_{\text{crude}} = x \cdot \left\{\max\left(0, \min\left(x, \frac{1}{2}\right)\right)\right\} \tag{5}$$

$$\widehat{f}^{\text{GeLU}}_{\text{crude}} = \frac{x}{1.702} \cdot \left\{\max\left(0, \min(\frac{x}{1.702}, \frac{1}{2})\right)\right\} \tag{6}$$

We show the the crude approximation error $\texttt{ApproxError}(x)$ for sigmoid, GeLU and SiLU in Figure 5 and draw attention to the following observations:

- $\texttt{ApproxError}(x)$ are non zero beyond a certain range. However outside this range $\texttt{ApproxError}(x)$ is zero for all $f(x)$.
- $\texttt{ApproxError}(x)$ is symmetric around Y-axis which means we can get away by approximating $\texttt{ApproxError}(x)$ for only $x \in [-5, 0]$ instead of trying to approximate for the whole range $[-5, +5]$.
- Finally note one can readily realize that $\texttt{ApproxError}(x)$ are much easier to approximate using piece-wise polynomials in contrast to original activation functions $f(x)$ as shown in Figure 2.

Remark: One of the implications of such handcrafted $\widehat{f}_{\text{crude}}(x)$ which is required by our scheme as input before designing MPC friendly piece-wise polynomials it that is not evidently clear how to design one for other complex activation functions such as Mish, and ELU. We address this limitation in Section **??**.

4.2 Estimating $P(x)$

Unlike prior works on generating MPC friendly approximation of non-linear functions [11, 30, 36], our scheme's approximation is input density aware. This means our scheme will focus more on minimizing errors for those areas where input density is high. As we will see, this helps us to ensure a low approximation error when errors are summed over for a fixed number of polynomials. Since $f(x)$ is evaluated obliviously (i.e., over encrypted inputs) it is counter intuitive and seems infeasible to have a prior over the input of x without observing client input. To realize this unintuitive approach, we leverage an unlikely techniques used in DNNs models for stability and fast training "batch normalization" (also described in Section 2.1.2). Since during inference phase, before input is forwarded to the activation layer, the inputs are normalized by mean and variance computed during the training phase. As a result, we can have density estimation over the input $P(x)$ to the activation function and optimize each activation functions based on their own input density calculated from the mean and variance.

Remark: This also requires model owners to keep these MPC-friendly approximations secret from client separating our work from previous work where MPC-friendly approximation can be public as they are density unaware. Our designed MPC-friendly approximation is density aware can essentially leak the batch mean and variance of private training data to clients if made public.

4.3 Interpolating $\texttt{ApproxError}(x)$

After we have designed $\widehat{f}_{\text{crude}}(x)$ and have a $P(x)$ over the input to the complex activation $f(x)$, the next task for our scheme is to approximate $\texttt{ApproxError}(x)$ efficiently. We do this approximation via a set of N polynomials of degree d covering the area $[x_s, x_e]$ as showin in Figure 1. First we set a value for n, the number of points to consider for approximation of $\texttt{ApproxError}(x)$. Generally while approximating a function via a set of N piece-wise polynomials covering a domain $[x_1, x_2, \cdots, x_n]$, the polynomials are equally spaced with each polynomial i approximating the domain $[x_i, x_i + \frac{(x_n - x_1)}{N}]$. However, in our case we want our interpolation to be density $P(x)$ aware — that is we will have more piece-wise polynomials for those regions $[x_i, x_j]$ where $P(x)$ is high. To realize this we calculate a density-aware approximation error threshold δ_{thr} (line #1). Then whenever we are considering if we want to

dedicate a new piece-wise polynomial to approximate a region $[x_i, x_j]$, we calculate the density aware approximation error δ_{crude} as shown in line #5 and Figure 2. If $\delta_{\text{crude}} > \delta_{\text{thr}}$ (line #6), then we approximate $\texttt{ApproxError}(x)$ for the region $[x_i, x_j]$ then we dedicate a new piece-wise polynomial $\text{poly}^d_{(x_s,x_i)}$ of degree at most d to approximate $\texttt{ApproxError}(x)$ for a region $[x_i, x_j]$ (line #7). Intuitively δ_{thr} signifies the approximation error we can tolerate before dedicating a new. Lastly, we explored two widely used interpolation methods Chebyshev and BSpline [] for finding $\text{poly}^d_{(x_s,x_i)}$ as shown in Figure 3.

5 EXPERIMENTAL EVALUATION

We run experiments to show that our scheme is able to generate MPC-friendly versions of three popular and complex and activation functions $f(x) \in [\text{sigmoid}, \text{SiLU}, \text{GeLU}]$ with negligible loss in accuracy and low latency. To do this we conducted extensive experiments answering the following questions.

- Section 5.2.1: To what extent accuracy degrades if we use our scheme generated MPC-friendly version $\widehat{f}_{\text{improved}}(x)$ over the actual complex activation functions $f(x)$ for image classification?
- Section 5.2.2: How does our scheme compares with recipes from prior works proposing MPC-friendly approximation of complex non-linear activation functions [16, 30, 36] or non-linear functions in general [11] for image classification?
- Section 5.3: Lastly, how much delay and loss in the quality of the prediction is added when we use our scheme generated MPC-friendly version $\widehat{f}_{\text{improved}}(x)$ for three real-time prediction tasks which are object detection, machine translation, and face anti-spoofing detection?

5.1 Implementation details

We implemented our scheme in `Python` with $\approx 3K$ Loc. Here we highlight some implementation challenges of our scheme as presented in Figure 1, and how we addressed them briefly.

- Number of points n to consider: We need to consider a reasonable number of points $[x_1, x_2, \cdots, x_n]$ for interpolation. A small n may resulted in poor $\widehat{f}_{\text{improved}}$ whereas a too large n may incur high latency for during prediction. We empirically evaluated different values of n as show that in Figure **??** and observed beyond $n \geq 1,000$ approximation error nor accuracy improves. Therefore for all of our experiments, we choose $n = 1,000$ points for interpolation.
- Interpolation: We explored two interpolating techniques, Chebyshev, and B-Splines widely used and known to produce accurate interpolation with low approximation error []. We used `numpy.polynomial.chebyshev` and `scipy.interpolate.` for interpolating polynomials respectively of degree $\leq d$.
- Extracting mean and variance during inference: One of the requirements of designing $P(x)$ is to approximate it using the mean and variance of each batch after the DNN model is trained. To extract the mean and variance of each layer we use the `model.running_mean` and `model.running_var` attributes available in Pytorch.

5.2 Image classification

To demonstrate the inference accuracy is negligible for secure inference using our scheme, we consider four state-of-the-art image classification models i) 3-layer fully connected networks (FCN) [], ii) 5-layer convolutional neural network (ConvNet) [], iii) 12 layer Residual neural network (ResNet) [] and iv) its scaled version EfficientNetB0 []. We highlight the description of these four models in briefly in Appendix A. Then we train these models on four widely used image classification datasets These four datasets which are MNIST, CIFAR-10, CIRF100 and ImageNet-1K have increasing complexity, size, and richness. Note while training we do not make any changes to the original parameters of the models as proposed in the literature such as architecture of the model, number of epochs, learning rate, optimizer etc.

We train the models for three complex activation functions $f(x) \in [\text{sigmoid}, \text{SiLU}, \text{GeLU}]$. During inference, We first run prediction using $f(x)$ and call it plaintext accuracy since using $f(x)$ in such a way requires plaintext access to clients input x. To record the loss in prediction accuracy if x were encrypted, we replace these complex activation functions $f(x)$ with their MPC-friendly counterpart and call it secure accuracy. Finally, we compare secure inference accuracy with plain text accuracy to measure the loss in prediction accuracy we have to endure for secure inference.

5.2.1 Comparing loss in accuracy for our scheme . We present the results for secure accuracy and plaintext accuracy for different models trained on widely used image datasets in Table 1. When we compare secure accuracy of our scheme with plaintext accuracy, we observed that

5.2.2 Comparison to related work. We compare our scheme with three other approaches proposed in the literature for generating MPC-friendly version of nonlinear functions [11] or specific activation functions [16, 30]. First we consider NFGen a recently proposed method for generating MPC-friendly version of a given nonlinear functions [11]. Their approach is more general compared to ours. However our designed techniques are specifically optimized for nonlinear activation functions used in DNN. Therefore when we adopted their recipe to generate MPC-friendly version of sigmoid, GeLU, and SiLU and switched their a MPC-version version during inference, we observed it suffers from significant loss in accuracy. Next, we compare it with another recent work Iron [16] that proposed a MPC-friendly receipe of GeLU activation function. We observed for image classification their loss in accuracy is still high. Lastly we consider Minion [30] that proposed MPC friendly version of sigmoid activation function which during inference does not show a significant loss inaccuracy for logistic regression and 3 layer DNN model trained on MNIST dataset. However, for DNN models trained on challenging datasets, their accuracy degrades significantly. Note that for a fair comparison with our work, we only adopted the recipe for generating MPC-friendly version of the complex activation function in isolation and did not adopt the secure matrix multiplication methods proposed in Iron and Minion approach. We can essentially adopt a similar secure matrix multiplication methods and boost up the secure accuracy of our scheme.

Dataset	Model	Activation Function	Accuracy plaintext	Accuracy Secure NFGen [11]	Iron [16]	Minion [30]	Our scheme
MNIST	FCN	sigmoid	92.42%	72.99%	✗	64.32%	92.39%
		SiLU	94.12%	86.83%	✗	✗	94.03%
		GeLU	94.53%	83.12%	82.39%	✗	93.97%
CIFAR-10	ConvNet	sigmoid	91.17%	86.77%	✗	90.62%	90.12%
		SiLU	90.59%	81.05%	✗	✗	91.83%
		GeLU	91.24%	61.05%	87.05%	✗	91.05%
CIFAR-100	ResNet	sigmoid	92.02%	91.59%	✗	91.59%	91.59%
		SiLU	93.33%	93.02%	✗	✗	93.02%
		GeLU	93.01%	92.66%	92.66%	✗	92.66%
ImageNet-1k	EfficientNetB0	sigmoid	95.45%	71.95%	✗	81.95%	91.95%
		SiLU	94.12%	93.95%	✗	✗	93.95%
		GeLU	92.34%	91.95%	91.95%	✗	91.95%

Table 1: Comparison of plaintext accuracy and secure accuracy for different DNN models train on image classification datasets for three activation functions. We put a ✗ mark for secure accuracy whenever the corresponding approach does not propose a MPC-friendly version of the corresponding activation function.

5.3 Real time secure inference

To investigate how much delay and loss in the quality of the prediction is added when we use our scheme generate MPC-friendly version $\widehat{f}_{\text{improved}}(x)$ for secure inference, we experimented with three real-time prediction tasks related to computer vision and natural language processing. Specifically, these tasks are 1) sentence translation, 2) object detection, and 3) face anti-spoofing detection used in face base user authentication. There are three reasons that motivated us in investigating these abovementioned tasks. Firstly the inference time — the time it takes to get the prediciton — is very important for these tasks in contrast to other tasks where client can wait for some time to get the prediciton result (e.g., secure medical analysis). Secondly the prediciton quality here is measured via number of scores instead of simple accuracy measure as with image classification. Lastly and most importantly for these three tasks the client inputs are private. Hence secure inference will play a crucial role to wide deployment of such services. Now we will briefly discuss these three real time tasks and then compare the results in terms of secure inference time and loss in prediction quality with plaintext inference time and prediction quality.

Sentence translation. Sentence translation in real-time is one of the popular and common features provided as a service by companies such as Google[6], Microsoft[7], Apple[8]. State-of-the model for sentence translation is attention-based transformer architecture [45]. Prior arts have shown that SiLU activation function outperforms or matches other baselines for sentence translation [40]. Therefore, in this task, we take 12 layers transformer architecture-based model and train it on standard WMT 2014 English-to-German translation datasets [7] for SiLU activation function in the feed-forward layer of the transformer model. The quality of sentence translation is measured via BLEU, and PPL scores between the original English to translated German language.

Object detection Real time object detection is another demanding prediction-based service currently provided as paid APIs by Microsoft [9], Amazon[10]. we train YOLO [44] — a state-of-the-art model for objection detection — on challenging common objects in context (MS-COCO) [29] dataset with GeLU activation function. The motivation behind using GeLU activation functions is that prior works have shown it is one of the best performing activation function for object detection. The quality of the detector is measured via AP score.

Measuring loss in prediction quality and inference time. Similar to our previous experiments in Section 5.2, we train the abovementioned state-of-the-art models corresponding to these three tasks for three activation function sigmoid, SiLU, and GeLU]. During inference, We first score prediction quality using $f(x)$ and call it plaintext inference score since achieving such scores would require having access to client's private input x. To measure the loss in prediction accuracy if x were encrypted for secure inference, we replace these complex activation functions $f(x)$ with their MPC-friendly counterpart and call it secure inference score. To measure the secure inference time, we assume the linear components of the DNN involving already MPC friendly operations $+, \times, >$ can be evaluated by any existing MPC libraries in the literature. Therefore to benchmark the latency in prediction time for secure accuracy we count the total number of MPC-friendly operations and multiply that with the unit time, it would take to perform those number of operations of the same message size under different threat models (semi-honest, malicious) and for the two-party and three-party scenario. To measure inference time when two parties are present (i.e., client and server) under semi-honest threat model, we used MP-SPDZ [25] — one of the popular efficient MPC libraries. When

[6] https://translate.google.com/
[7] https://www.microsoft.com/en-us/translator/
[8] https://support.apple.com/guide/iphone/translate-text-voice-and-conversations-iphd74cb4501/ios
[9] https://learn.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection
[10] https://docs.aws.amazon.com/sagemaker/latest/dg/object-detection.html

Task	Model	Dataset	Activation function	Scoring metric	Plaintext inference score	Secure inference score
Sentence translation	Transformer	WMT-14	SiLU	BELU	26.86	26.70
				PPL	83.25	84.90
Object detection	YOLOv4	MS-COCO	GeLU	AP^{Val}	33.7%	32.3%
				AP^{Val}_{50}	37.4%	37.4%
				AP^{Val}_{75}	22.9%	20.9%
				AP^{Val}_{s}	62.3%	62.3%

Table 2: Comparison of prediction quality between plaintex inference and secure inference for three real time prediction tasks.

Task	Input Size	Plaintext inference time	Secure inference time	
			2PC	3PC
Sentence translation	1.5K	7.8s	10.8s	8.3s
Object detection	1024	1.2s	3.2s	2.2s

Table 3: Comparison of prediction time between plaintext inference and secure inference for two real-time prediction tasks.

there are more than two parties present, as the scenario introduced in Araki et. al [2], we used another popular MPC library ABY^3 [35]. This time too we assume semi-honest threat model and honest majority among the three parties. That being said any MPC libraries [17] (e.g., EMP-toolkit, MOTION, and more) supporting $+, \times, >$ can be used for benchmarking.

5.3.1 How much prediction quality degrades for secure inference? We show the loss in prediction quality in Table 2 for all three prediction services describe above where quality is measured under different metrics.

5.3.2 How much delay is added for secure inference? We show the loss in prediction quality in Table 3. For this experiment to measure the delay in secure inference

6 CONCLUSION

To protect client's privacy, we design a scheme that enables deep neural networks (DNN) to do prediction on encrypted input by generating MPC-friendly version of complex activation functions (AF) used in DNN. Our scheme of generating MPC-friendly version of complex AF has been specially optimized so that when we replace our MPC-friendly function with their complex AF counterpart during prediction it does not cause significant loss in prediction quality or latency. We conduct extensive experiments and show the applicability of our approach for a number of complex tasks involving state-of-the-art DNN models used in computer vision, natural language processing and face anti-spoofing detection and trained on challenging datasets.

REFERENCES

[1] Amit Agarwal, Stanislav Peceny, Mariana Raykova, Phillipp Schoppmann, and Karn Seth. Communication efficient secure logistic regression. *Cryptology ePrint Archive*, 2022.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016.

[3] Shayan Aziznejad and Michael Unser. Deep spline networks with control of lipschitz regularity. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3242–3246. IEEE, 2019.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Pakshal Bohra, Joaquim Campos, Harshit Gupta, Shayan Aziznejad, and Michael Unser. Learning activation functions in deep (spline) neural networks. *IEEE Open Journal of Signal Processing*, 1:295–309, 2020.

[6] Pakshal Bohra, Dimitris Perdios, Alexis Goujon, Sébastien Emery, and Michael Unser. Learning lipschitz-controlled activation functions in neural networks for plug-and-play image reconstruction methods. In *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*, 2021.

[7] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, et al. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation*, pages 12–58, 2014.

[8] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. SIMC: ML inference secure against malicious clients at Semi-Honest cost. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1361–1378, Boston, MA, August 2022. USENIX Association.

[9] Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, 2020.

[10] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[11] Xiaoyu Fan, Kun Chen, Guosai Wang, Mingchun Zhuang, Yi Li, and Wei Xu. NFGen: Automatic Non-Linear Function Evaluation Code Generator for General-Purpose MPC Platforms. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 995–1008, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Frederic B Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *The Journal of Symbolic Logic*, 9(2):49–50, 1944.

[13] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.

[14] Karthik Garimella, Nandan Kumar Jha, and Brandon Reagen. Sisyphus: A cautionary tale of using low-degree polynomial activations in privacy-preserving deep learning. *arXiv preprint arXiv:2107.12342*, 2021.

[15] 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 *International conference on machine learning*, pages 201–210. PMLR, 2016.

[16] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems*, 2022.

[17] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE symposium on security and privacy (SP)*, pages 1220–1237. IEEE, 2019.

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[19] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.
[20] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.
[21] 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.
[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
[23] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.
[24] Mahimna Kelkar, Phi Hung Le, Mariana Raykova, and Karn Seth. Secure poisson regression. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 791–808, 2022.
[25] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590, 2020.
[26] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model Extraction Warning in MLaaS Paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, page 371–380, New York, NY, USA, 2018. Association for Computing Machinery.
[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
[28] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2201–2218, 2021.
[29] TY Lin, M Maire, S Belongie, J Hays, P Perona, D Ramanan, P Dollár, CL Zitnick, et al. Microsoft coco: Common objects in context. ineuropean conference on computer vision 2014 sep 6 (pp. 740-755).
[30] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 619–631, 2017.
[31] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.
[32] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.
[33] 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.
[34] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 2019.
[35] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 35–52, New York, NY, USA, 2018. Association for Computing Machinery.
[36] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.
[37] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.
[38] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, and Michael Unser. Approximation of lipschitz functions using deep spline neural networks. *arXiv preprint arXiv:2204.06233*, 2022.
[39] George-Liviu Pereteanu, Amir Alansary, and Jonathan Passerat-Palmbach. Split he: Fast secure inference combining split learning and homomorphic encryption. *arXiv preprint arXiv:2202.13351*, 2022.
[40] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
[41] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342, 2020.
[42] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. {XONN}:{XNOR-based} Oblivious Deep Neural Network Inference. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1501–1518, 2019.
[43] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pages 707–721, 2018.
[44] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, and Alexander Wong. Fast yolo: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*, 2017.
[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
[46] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019.
[47] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.
[48] Yu-Dong Zhang, Chichun Pan, Junding Sun, and Chaosheng Tang. Multiple sclerosis identification by convolutional neural network with dropout and parametric relu. *Journal of computational science*, 28:1–10, 2018.

A DNN MODEL ARCHITECTURE