Compact: Approximating Complex Activation Functions for Secure Computation

Mazharul Islam*, Sunpreet S. Arora†, Rahul Chatterjee*, Peter Rindal†, Maliheh Shirvanian‡
* University of Wisconsin—Madison, † Visa Research, ‡ Netflix

ABSTRACT

Secure multi-party computation (MPC) techniques can be used to provide data privacy when users query deep neural network (DNN) models hosted on a public cloud. State-of-the-art MPC techniques can be directly leveraged for DNN models that use simple activation functions such as ReLU. However, these techniques are ineffective for the complex and highly non-linear activation functions used in cutting-edge DNN models.

We present Compact, which produces piece-wise polynomial approximations of complex activation functions that can be used with state-of-the-art MPC techniques. Compact neither requires nor imposes any restriction on model training and achievesnear-identical model accuracy. We design Compact with input density awareness and use an application specific simulated annealing type optimization to generate computationally efficient approximations of complex activation functions. We extensively evaluate Compact on four different machine-learning tasks with DNN architectures that use popular complex activation functions SiLU, GeLU, and Mish. Our experimental results show that Compact incurs negligible accuracy loss while being 2×—5× faster than state-of-the-art approaches for DNN models with large number of hidden layers. Our work accelerates easy adoption of MPC techniques to provide user data privacy even when the queried DNN models consist of a number of hidden layers and complex activation functions.

1 INTRODUCTION

Deep neural networks (DNNs) based inference services are being increasingly adopted in various emerging applications, such as early disease discovery from personal health records [76], personalized product recommendations [13], media translations [12], image recognition [3], and even biometric authentication [18]. Trained DNN models are typically hosted on a cloud server for applications or users to query for inference tasks. These services however can pose serious privacy concerns. For instance, users are required to share their facial images with an online service hosting a face recognition DNN model. Indeed due to such privacy concerns, the Internal Revenue Service (IRS) removed the identity verification service based on facial recognition [18].

DNN models used for inference cannot be transferred to the client devices because they can be proprietary and trained on private training data such as users' medical records [23, 43]. Clients also would like to avoid sharing their private data with the model hosting server. This problem is generally referred to as *secure inference*, where a client can obtain the inference results on their private input without sharing it with the server, nor learning anything about the DNN model parameters. The secure multi-party computation (MPC) is a promising approach to solve the secure inference problem [81]. However, a key challenge is computing the non-linear activation functions (AFs) efficiently. Indeed, studies have shown that AFs are the bottleneck — compared to linear layers — for performing secure inference [24, 25, 36, 56, 68].

Prior works [11, 14, 26, 47, 68, 69] have provided several solutions for secure inference for DNN models with ReLU AF — a relatively simple non-linear AFs. However, lately, ML researchers are favoring more complex and highly non-linear AFs such as SiLU, GeLU, Mish for new ML applications. State-of-the-art secure inference protocols are either unsuitable or inefficient for DNN models trained over these complex AFs.

Complex AFs can be approximated using piece-wise polynomials for efficient computation in MPC frameworks, as shown in recent works [48, 51, 60]. However, a key limitation of these approaches is that they incur high accuracy loss compared to plaintext ("not secure") inference. Fan et al. recently proposed NFGen [21], that can generate MPC-friendly polynomial approximation of a variety of non-linear functions used in scientific domains without introducing significant accuracy loss. Although this approach is generic and can be used to approximate complex AFs, doing so with NFGen incurs significant performance overhead (as we show in Section 5.4).

This is because generating approximations of complex AFs that have both negligible accuracy loss, and performance overhead requires carefully searching for optimal parameters used in the approximation process. Conservatively setting these parameters, as done in NFGen, to generate an approximation that does not introduce significant accuracy loss will in turn increase the computational overhead. One may settle for parameters that yield an imprecise approximation of AFs to improve the speed of secure inference using NFGen, but it will end up degrading the accuracy of the DNN model. Swapping out the complex AF with ReLU increases accuracy loss, and might require retraining or fine-tuning, which are computationally expensive for large DNN models.

We devise a new approach to approximate complex AFs that does not degrade inference accuracy even for DNN models with many hidden layers. We do so without requiring any retraining of the model or change in the model architecture.

There are two main challenges to achieve an improved approximation of AFs. First, due to input normalization, most of the inputs to AFs are around zero, where complex AFs are highly non-linear. Chabanne et al. [10] observed that for a nine-layered DNN model normalization pushes 99.73% of the input values to the ReLU AF

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–17

https://doi.org/XXXXXXX.XXXXXXX

between $[-3, 3]$. Second, the approximation approach needs to balance the trade-offs between performance overhead and inference accuracy loss carefully.

To handle these two challenges, we incorporate the observation that in state-of-the-art DNN models, inputs to complex AFs are normalized, into our approximation generation process. Such normalization gives a way to estimate the input probability density to the complex AF as the majority of the normalized inputs to the complex AF would fall into specific places near the region close to zero with high probability — while a small portion will fall into places in regions away from zero with low probability. We hypothesize that taking this observation into account will help mitigate the cumulative impact of errors introduced by MPC-friendly approximations from one layer to subsequent layers of a deeper or wider DNN model. While prior work [10] has used this observation for generating fully homomorphic encryption (FHE)-friendly approximations of ReLU AF, their proposed approach of using a single polynomial generated via "least square fit" would not work well for MPC-friendly approximations of complex AFs when DNN models have a high number of hidden layers [55].

We use the Chebyshev sequence-based interpolation [53] for piece-wise polynomial approximation, which is shown to provide better approximation for non-linear functions involving operations, including e^{-x}, $\ln(x)$, $\tanh(x)$, $1/x$, etc., as it is the case with complex AFs [55, Table 5.2]. [1]

Piece-wise polynomial approximation can be parameterized by the degree of the polynomial (k), the number of pieces (m), and the Ring in which the MPC will be executed ($\mathcal{R}$). We establish a procedure to find a better tradeoff between accuracy loss and computational overhead. using application-specific heuristic that dynamically adjusts these parameters to find a desirable piece-wise polynomial. Specifically, we first pose this problem as a constraint optimization problem (COP) by setting a constraint of the maximum accuracy loss, say ν, that a practitioner can tolerate. Then we search for a $\langle \mathsf{m}', \mathsf{k}', \mathcal{R}' \rangle$ that yields an approximation that has the lowest inference time under the constraint that accuracy loss is below ν. We base our searching heuristic on simulated annealing (SA) which is a popular framework to solve COP.

Concretely, we start with an initial solution with high $\langle \mathsf{m}_0, \mathsf{k}_0, \mathcal{R}_0 \rangle$ so that it yields an approximation that has accuracy loss $\leq \nu$, but not necessarily the lowest inference time. As a result the initial choice of $\langle \mathsf{m}_0, \mathsf{k}_0, \mathcal{R}_0 \rangle$ may result in an approximation having pronounced inference time yielding an imbalance between performance and accuracy. To fix this, we randomly make local adjustments to explore adjacent solutions of $\langle \mathsf{m}_0, \mathsf{k}_0, \mathcal{R}_0 \rangle$, and continue moving towards a solution $\langle \mathsf{m}', \mathsf{k}', \mathcal{R}' \rangle$ under the SA framework that reduces performance overhead and accuracy loss remains less than ν — for a fixed number of iterations.

We carefully incorporate a number of application-specific techniques into the heuristic to avoid getting stuck on local optimal $\langle \mathsf{m}', \mathsf{k}', \mathcal{R}' \rangle$. For example, we find the approximation error threshold — an important component of the heuristic — via binary search instead of settling for a fixed value as prior work [21] (Section 4.2.3).

[1] Although NFGen also used Chebyshev interpolation, they did not consider input normalization to improve accuracy.

Furthermore, we introduce a DNN-specific modification to enhance the performance efficacy (Section 4.2.5).

We implement Compact and perform extensive experiments using four different state-of-the-art DNN models with many hidden layers on diverse classification tasks. We find that Compact and NFGen [21] incur negligible accuracy loss compared to existing approaches [48, 51, 67] (Section 5.3). Then, to compare performance overhead between Compact and NFGen, we incorporate their generated MPC-friendly approximation of complex AFs to two state-of-the-art secure inference MPC libraries ABY3 [58] and CryptFlow2 [68], and measure average inference time. Our experiments reveal that our DNN model-specific optimizations make Compact 2×–5× computationally more efficient than NFGen [21] — for DNN models having a high number of hidden layers, all while maintaining negligible accuracy loss. We have released Compact as an open source project [39].

Summary. Our contributions are as follows:

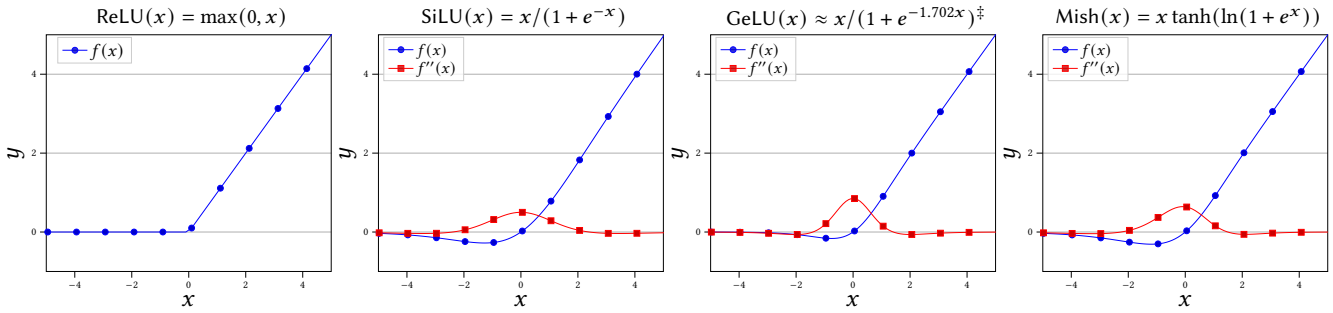
- We present Compact, a scheme that can generate MPC-friendly piece-wise polynomial approximations for complex non-linear AFs. The generated approximation is generic and can be easily incorporated into state-of-the-art multi-party computation scenarios (Section 4.1).
- The approximation technique used in our scheme is input density aware and accurately approximates regions with high input probability density while coarsely estimating regions with low input probability density (Section 4.2).
- We propose a new searching heuristic based on simulated annealing framework to find parameters that dynamically adjust an approximation that have balance performance overhead and accuracy loss (Section 4.3).
- We conduct extensive experiments and show that Compact generated MPC-friendly approximation of complex AFs have both negligible inference accuracy loss than other DNN-specific approaches [48, 51, 67], and 2×–5× faster than NFGen [21] (Section 5).

2 BACKGROUND AND RELATED WORK

This section summarizes relevant background and prior work from deep neural networks (Section 2.1) and cryptographic techniques developed for solving the secure inference problem (Section 2.2).

2.1 Deep Neural Network Preliminaries

Activation Functions (AFs). AFs are used for adding non-linearity to the learning process and play a major role in enhancing the training capabilities and accuracy of the DNN models. Many contemporary models use ReLU AF, which makes a hard gating decision based on the input sign (Figure 1). Despite being theoretically simple, ReLU provides remarkably faster convergence and performance in practice [45, 61]. However, ReLU outputs a value of zero whenever the input is negative, and as such, the neural network loses a certain amount of valid information as soon as inputs become



† Second derivatives $f''(x)$ equal to zero indicates that linear polynomials can easily approximate the function.
‡ More accurate version is GeLU$(x) = 0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$.

Figure 1: Complex activation functions (AFs) we focus in our work $f(x) \in \{$SiLU, GeLU, Mish$\}$ and their second derivatives $f''(x)$. These AFs are hard to approximate accurately in regions close to zero where $f''(x) > 0$. We argue this is especially problematic for DNN models as the majority of the input to the complex AF falls to the region that are hard to approximate accurately (i.e., close to zero) due to normalization (Figure 2). In contrast, ReLU(x) AF can be precisely approximated with only two simple polynomials $\{f_1, f_2\}$ which are $f_1(x) = 0$ when $x < 0$ and $f_2(x) = x$ when $x \geq 0$.

negative. This drawback prompted ML communities to develop complex AFs, overcoming the limitations of ReLU.

Complex AFs. In recent years, a range of complex AFs, such as SiLU [19], GeLU [31], and Mish [57], have emerged surpassing the performance of ReLU in state-of-the-art DNN models applied across computer vision, natural language processing, and reinforcement learning applications. These AFs as shown in Figure 1, are smooth and continuous, can handle small weight changes, and aid in effectively regularizing DNN models. For example, Hendrycks et al. [31] empirically illustrated the robustness of GeLU-trained DNN models against noisy inputs, often surpassing the accuracy of ReLU-trained models. Ramachandran et al. [66] used automatic search techniques to uncover SiLU (also called Swish). This complex AF improved image classification accuracy of Inception-ResNet-v2 model by 0.6% and by 0.9% of Mobile NASNET-A model by simply substituting it with ReLU. Misra et al. [57] proposed the self-regularized AF Mish that exhibits superior performance compared to AFs for YOLOv4, ResNet models.

Hence, complex AFs offer a compelling advantage in building better-performing models in terms of convergence and classification accuracy when compared to ReLU. Unfortunately, unlike ReLU, which is relatively easy to compute for secure evaluation, these complex AFs exhibit a higher degree of non-linearity near the region close to zero as shown in Figure 1. This makes their use with existing MPC techniques challenging. In this work, we address this limitation by designing an MPC-friendly version of these three complex AFs. We refer more interested readers to Appendix A for additional details on other complex AFs used in neural networks that lie outside the scope of this work.

Batch normalization. Batch normalization (BN) is used to address *internal covariance shift* problem in neural networks — which happens when a layer's input distribution changes abruptly due to its dependency on previous layers [38]. BN lends stability to the training process by reducing dependence on initial parameter selection, requiring a lower learning rate, and number of epochs. BN

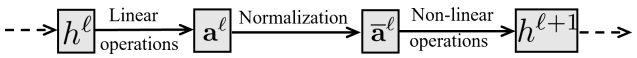


Figure 2: The output of the linear operations ($\mathbf{a}^\ell$) are normalized to $\overline{\mathbf{a}}^\ell$ using Equation (2) before they are forwarded for applying non-linear operations involving complex activation functions (AFs).

is performed on the outputs of the linear transformations, and normalized outputs are forwarded to non-linear AFs. Thus, non-linear AFs receive normalized inputs. Figure 2 illustrates BN process for ℓ^{th} layer where input to the linear operations is h^ℓ and output is $\mathbf{a}^\ell = w^{\ell T} h^l$. Assume $\mathbf{a}^\ell = \left(a_1^\ell, a_2^\ell, \cdots, a_d^\ell\right)$ is d-dimensional. If the population mean and variance are $\mathsf{E}[\mathbf{a}^\ell], \mathsf{Var}[\mathbf{a}^\ell]$ respectively, then $\mathbf{a}^\ell$ is normalized to $\overline{\mathbf{a}}^l$ using the following Equation (2) such that the probability distribution of $\overline{a}^l$ follows a normal distribution with zero mean and unit variance:

$$\overline{\mathbf{a}}_k^\ell = (\mathbf{a}_k^\ell - \mathsf{E}[\mathbf{a}_k^\ell])/\mathsf{Var}[\mathbf{a}_k^\ell] \quad (2) \qquad (2)$$

BN is widely used in state-of-the-art DNN models to calibrate the input to the non-linear AFs during both training and inference phases. This makes BN a good estimator of the input density to complex AFs in DNN models during inference. Our scheme leverages this estimation to improve the inference accuracy of the generated MPC-friendly approximations.

2.2 Secure Inference for DNN models

State-of-the-art MPC techniques enable computation on encrypted data and have been used to address the secure inference problem. Generally, a client encrypts their input and sends the encrypted input to a cloud service. The cloud service performs inference using trained DNN models over the encrypted input. Typically, MPC techniques are optimized for linear transformations (e.g., addition, matrix-vector multiplications, etc.). Therefore, computing

non-linear operations involved in secure inference (e.g., non-linear AFs) is one of the main challenges.

ReLU specific secure inference. Given the popularity of ReLU in practical deployments of DNNs, recent research has mostly focused on the use of ReLU. Early two works [10, 26] in this area generate Fully Homomorphic Encryption (FHE) friendly approximation of ReLU AF for secure inference. Recent work focuses on MPC friendly approximation of ReLU [11, 35, 68, 77, 78].

For example, Rathee et al. [68] propose a novel 2PC protocol for secure comparison and division for efficient evaluation of ReLU in semi-honest settings. Follow-up works extend this protocol to the malicious client threat model [11, 35]. However, ReLU specific optimizations proposed in the aforementioned methods do not generalize to other complex AFs. Another set of methods uses Garbled Circuits (GC) for secure evaluation of ReLU [41, 47, 56, 70]. However, communication overhead limits its applicability to shallow DNN models (less than seven layers). It is challenging to generalize these methods to wide DNN models that use complex AFs other than ReLU for secure inference.

A different approach for computing non-linear AFs efficiently in the encrypted domain is by restricting the way DNN models are trained. For example, Riazi et al. [69] leverage GC based protocol for secure inference on binary neural networks (BNN). However, retraining and pruning proprietary models with these restrictions could be costly and oftentimes practically infeasible. Imposing such limitations on the training process can also impact the performance of DNNs in practice. Pereteanu et al. [65] introduce the notion of partially private DNN models such that the middle part of the model is sent in plaintext to clients to reduce communication overhead. However, in practice, cloud service providers would want to keep their full part of the DNN model secret lest revealing any part of the property model leaks sensitive information, resulting in severe business consequences.

In summary, while many promising works [63] have focused on secure inference for ReLU-based DNNs, our work focuses on novel complex AFs that have been shown to outperform ReLU and are getting traction in the ML community.

Secure inference for other non-linear AFs. A common approach for secure inference involving non-linear AFs is by approximating them with low-degree polynomials. These polynomials are easy to compute for MPC frameworks and thus are MPC-friendly. The challenge is not to degrade the inference accuracy, as the approximation error can cause incorrect results. Delphi [56], for example, runs a planner that balances which AF can be replaced with low-degree polynomials without introducing too many inaccuracies and achieving a significant communication benefit. CryptoNet [26] CryptoDL [32], MiniONN [51] also, use similar ideas for approximating non-linear AFs. However, they are application-specific, and switching to another application degrades accuracy significantly [24] due to small errors getting propagated resulting in numeric instability. In addition, MiniONN [51] is heavily focused on sigmoid AF — which is essential for logistic regression models.

However, as we will show in Section 5.3, when we use their recipe for generating MPC-friendly approximation of the complex AFs that we focus on in this work, the inference accuracy decreases drastically. Recently, Fan et al. [21] proposed NFGen, a technique capable of converting popular non-linear functions — used in scientific domains — to MPC-friendly ones. One may also choose to use this approximation-based approach to do the same for complex non-linear AFs. In fact, NFGen is the closest related work to ours. However, NFGen is not specifically customized for widely used complex AFs inside DNN models. Absence of such customized techniques makes NFGen computationally less efficient when we compare it with our scheme through extensive experiments (Section 5.4).

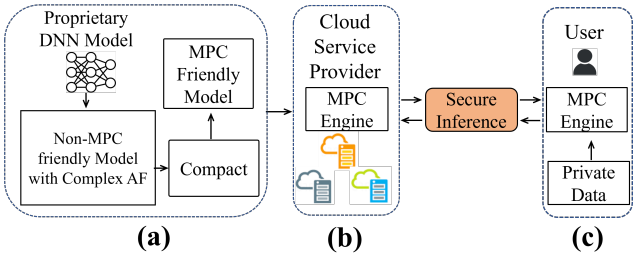


Figure 3: Secure inference in cloud-based deployment setting. (a) Proprietary DNN model trained over private data that is not MPC-friendly due to complex non-linear activation functions (AFs) (e.g., SiLU, GeLU, Mish). An MPC-friendly model is generated by replacing the complex AFs with their approximations using Compact. (b) Next, we generate n secret shares of the MPC-friendly model and distribute them with n computing servers (in this figure $n = 3$) on the cloud. (c) To get the inference result, the client gets the private input data from the user, generates shares of it and distribute these with the n servers. These servers on the cloud perform secure inference using an MPC engine and return the shares of the inference result to the client, and the client uses them to reconstruct the original inference result.

3 PROBLEM OVERVIEW & DESIGN GOALS

In this section, we first formulate the problem of secure inference and detail the threat model (Section 3.1). Then we describe the design goals we want to ensure while developing Compact (Section 3.2).

3.1 Problem Overview

Problem formulation. We refer to the server holding the DNN model by $\mathcal{S}_{\text{owner}}$. The DNN model consists of L layers, each comprising linear transformations and non-linear complex activation function (AF) F_{act}. In between linear and non-linear complex AF, batch normalization is also present. We assume a machine learning as a service (MLasS) [1] inspired scenario where weights $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_L]$ of all L layers of the model have already been trained, and the trained model is being used to provide cloud-based inference z over client (C) uploaded input $\mathbf{x}$ using Equation (3).

$$z := F_{\text{act}}(\mathbf{w}_L \cdot \cdots F_{\text{act}}(\mathbf{w}_2 \cdot F_{\text{act}}(\mathbf{w}_1 \cdot \mathbf{x}))) \tag{3}$$

The problem secure inference tackles is how to compute the above equation *obliviously* to satisfy the privacy needs of both C and $\mathcal{S}_{\text{owner}}$. This requires designing a system such that C knows

nothing about model weights $\mathbf{W}$ and $\mathcal{S}_{\text{owner}}$ learns nothing about $\mathbf{x}$; yet C can get the inference result z. Moreover, we need to achieve this privacy need with both negligible accuracy loss and reduced performance overhead.

Scenario Setup. Secure multi-party computation (MPC) techniques enable a set of mutually distrusting parties to compute a function over their private inputs without revealing the inputs to other parties. Most MPC platforms use a version of *secret sharing* [5]. A (t, n)-secret sharing scheme divides a secret input s into n shares, such that any $t-1$ of these shares reveal no information about s, whereas any t shares allow complete reconstruction of s. Based on this primitive, we consider the following scenario to solve the secure inference problem using MPC.

In our setup, there are two parties: the first one is $\mathcal{S}_{\text{owner}}$ who owns the trained model with weights $\mathbf{W}$. The second party is the user C who queries the model with their private data $\mathbf{x}$. In a typical secure inference system, a set of semi-honest computing servers help compute the secure inference. These servers do not have any input of their own but facilitate the secure computation procedure. Neither the model owner $\mathcal{S}_{\text{owner}}$ nor the client C trust these servers with their private inputs; however, they trust them to follow the protocol specified.

Secure inference is performed in two phases. In the first phase, $\mathcal{S}_{\text{owner}}$ locally generates n secret shares of their private data $\mathbf{W}$ as $[\![\mathbf{W}_1, \mathbf{W}_2, \cdots \mathbf{W}_n]\!]$, and distributes them amongst n computing servers over the network. After this phase, when the client C wants to query the secure inference service with their privacy-sensitive data $\mathbf{x}$, they need to generate n secret shares of it locally as $[\![\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]\!]$, and send these shares to the n computing servers. Finally, these n computing servers engage in an MPC protocol to securely compute Equation (3), and generate n secret shares of the result z. At the end of the protocol, t of these secret shares of z are sent to the querying user who can combine these n shares to construct the final result z. Following prior work we consider two settings based on the number of computing servers: (i) $n = 2$ [56, 60], and (ii) $n = 3$ [71, 78, 78]. In this study, we refer to them as 2PC and 3PC scenarios respectively.

Threat model and scope. In this work, the techniques we use do not apply any restriction on how the adversary $\mathcal{A}$ is modeled by the MPC scheme or the type of secret sharing being used by the MPC scheme. Henceforth we can inherit the security requirements of the underlying MPC scheme.

That being said, the majority of the existing works on secure inference assume that these n computing servers are semi-honest (i.e., adversaries who do not deviate from the protocol but try to learn as much information as possible as from the messages they receive) [63]. In practice, this can be achieved by placing these computing servers under the regulation of a trusted organization, and monitoring that they are following the MPC protocol. In our experimental evaluation, we adopt this threat model.

We note that our scheme does not guarantee protection of $\mathbf{W}$, and $\mathbf{x}$ against attacks such as training data poisoning [73], model inversion [23], adversarial examples [27], membership inference attacks [9], etc. One wishes to do so should employ defenses from existing literature, and whether MPC schemes can be leveraged to provide protections against such attacks is an open question as discussed further in Section 6.

Method	❶ Supp. cmplx. AF	❷ Supp. many HL	❸ Comp. w/ MPC libs	❹ Supp. any training proc.
CryptFlow2 [68]★	○	●	○	●
SIMC [11]★	◐	●	○	●
Cheeta [35]★	○	●	○	●
Delphi [56]★§	○	●	○	○
XONN [69]★	◐	●	○	○
SIRNN [67]	◐	○	○	●
MUSE [47]★	◐	●	○	●
SecureNN [77]★	○	●	○	●
FALCON [78]★	○	●	○	●
NFGen [21]	●	○	●	●
Compact (this)	●	●	●	●

● Yes; ◐ Unclear; ○ No

★ Support for large number of hidden layers (HLs) for these methods is experimentally validated for DNN models for ReLU.
§ Although Delphi uses GC to evaluate non-linear layers, the MPC-friendly square function used to replace ReLU is specific towards ReLU. Therefore, we do not consider it compatible with complex AFs.

Table 1: Comparison of related work with Compact.

Motivating example. One of the motivating realizations of secure inference scenarios can be in the medical domain as pictured in Figure 3. In particular, where a DNN model weights $\mathbf{W}$ has been trained by a trusted organization (e.g., National Institutes of Health) leveraging substantial computational resources and exclusive access to users' private health records. To preserve the privacy of the proprietary DNN model, NIH can generate secret shares of the model $\mathbf{W}$ and distribute them across n different semi-honest computing servers, possibly hosted by different hospitals. When patients submit their private health data $\mathbf{x}$, they can generate n secret shares and share them with the n different hospitals. In this way, the patient learns the final result without learning anything about $\mathbf{W}$ or revealing their private information $\mathbf{x}$ to any hospital.

Difficulty in computing non-linear AFs. A major bottleneck while running the MPC protocol is computing $F_{\text{act}}(x)$ securely shown in Equation (3). This is because $F_{\text{act}}(x)$ is non-linear, which consumes most of the communication and latency costs of the overall protocol execution, as illustrated by many prior works (e.g., Rathee et al. [68, Table 6]). Linear operations (i.e., matrix-vector multiplication) are less expensive comparatively.

3.2 Design Goals

While designing Compact, we want to ensure DNN model designers are not restricted to the set of AFs and model architectures that MPC platforms support. We distill four criteria for this and show how prior work on secure inference fail to satisfy one or more of these design goals in Table 1.

❶ Support complex AF. We want our scheme to be compatible with the majority of the DNN models used by inference services. Therefore, in this work, we do not use ReLU-specific optimizations. Majority of prior works are devoted to optimize ReLU and fail to satisfy this design goal [35, 56, 68, 77, 78]. Few works rely on garbled circuits (GC) to evaluate AFs, but experimental evaluations

are limited to ReLU AF [11, 47, 67, 69]. Therefore, it is unclear if these GC-based protocols can generalize to other complex AFs such as SiLU, GeLU, and Mish. We marked them as *unclear* in the first column of Table 1.

❷ **Supports large number of hidden layers.** The error introduced due to replacing F_{act} with its MPC-friendly approximation $\widehat{F}_{\text{act}}$ in Equation (3) can accumulate and possibly lead to a significant loss in accuracy as the number of hidden layers increases. Unfortunately, few prior works [48, 51, 67] that support complex AFs show significant accuracy loss for DNN models with a high number of hidden layers. NFGen, however, does not exhibit this accuracy loss as the number of hidden layers increases, but this negligible accuracy loss comes at the cost of paying high-performance overhead. We want our scheme to endure such accuracy loss as the number of hidden layers increases without increasing the performance overhead significantly.

❸ **Compatible with MPC libraries.** The secure inference procedure we develop should not only support a wide variety of AFs but also should be easy to implement. Implementations that require new cryptographic primitives for secure inference will be hard and slow to deploy. Therefore, in this work, we aim to design a scheme that can be implemented with generic MPC libraries currently in use. Our solution only requires secure addition, multiplication, and comparison operations. This would also allow seamless transitioning from inference service using ReLU based DNN models to complex AF-based DNNs. Prior works other than NFGen do not satisfy this design goal.

❹ **No restriction on training.** To handle accuracy loss with an increasing number of hidden layers, some prior works change the way DNN models are traditionally trained. For example, XONN requires restricting the weights of the DNN model to binary values (i.e., ±1); similarly, Delphi replaces certain AFs (i.e., ReLU) with a square function during training. We believe this type of restriction poses additional constraints for deployment of secure inference as existing DNN models are most likely trained without these restrictions, and the weight of the already trained DNN models must be adjusted (e.g., fine-tuning by applying these restrictions) to comply with these protocols. Therefore, we aim to design Compact without any restriction on the training process of the DNN models.

In summary, recent proposal in secure inference literature holds promise toward realizing secure inference; but they do not satisfy the above-mentioned generality, deployability, and scalability aspects important for realizing secure inference in practice. We aim to bridge this gap via our designed scheme Compact.

4 DESIGN OF COMPACT

In this section, we first give an overview of our scheme in Section 4.1. Then, we gradually detail our scheme in Section 4.2 and Section 4.3. We sketch our scheme in Figure 4 with a summary of used notations in Table 2.

4.1 Overview of Compact

Piece-wise polynomial approximation approach. Our scheme Compact follows the idea of approximating a complex activation function (AF) using a number of piece-wise polynomials. First, we observe that complex AF can be approximated easily using linear functions outside a certain range. (Fan et al. [21] made similar observations for sigmoid.) Therefore, we only need to approximate a small range of x values, say $[s, e]$. We will approximate F_{act} using a piece-wise polynomial function, with m pieces $[f_1, f_2, \cdots, f_m]$ defined as follows:

Symbol	Description of the symbol
$F_{\text{act}}(x)$	complex activation function we want to approximate.
$\widehat{F}_{\text{act}}(x)$	MPC friendly piece-wise polynomial approximation of $F_{\text{act}}(x)$.
$\mathcal{E}$	distance metric used to estimate the approximation error between $F_{\text{act}}(x)$ and $\widehat{F}_{\text{act}}(x)$
δ	maximum threshold for approximation error.
m	# of piece-wise polynomials used for approximation.
k	maximum degree of each of m piece-wise polynomials.
$\mathcal{R}_{\ell,d}$	ring of size ℓ is used in MPC library with last d bits representing the fractional parts.
$f_i(x)$	single polynomial approximating $F_{\text{act}}(x)$ between $[x_i, x_{i+1}]$
$[s, e]$	the interval over which we are trying to approximate F_{act}
$[\alpha, \beta]$	a continuous closed interval between α and β
P	probability distribution of the input to the activation function.

Table 2: Notations used in this paper.

$$\widehat{F}_{\text{act}}(x) = \sum_{i=0}^{m+1} I_i(x) \cdot f_i(x) \tag{4}$$

where $I_i(x) = 1$ if $x \in (x_{i-1}, x_i]$, and 0 otherwise, for all $i \in \{0, 1, \ldots, m+1\}$, $x_{-1} = -\infty$, $x_0 = s$, $x_m = e$, and $I_{m+1}(x) = 1$, if $x > e$, and 0 otherwise. The functions I_i define the pieces, and functions f_i define the polynomials. From Figure 1, it is easy to see that when $x \leq -5$ or $x \geq 5$, $F_{\text{act}}(x)$ becomes equal (or very close) to zero and x, respectively. As such, we can set $f_0(x) = 0$, and $f_{m+1}(x) = x$ by maintaining $s \leq -5$, and $e \geq 5$ for all complex AFs. For the other polynomials, we impose an additional constrain that f_i must be of degree k or less, $\forall i$ $\text{Deg}(f_i) \leq \text{k}$ as following.

$$f_i(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k \tag{5}$$

The above-mentioned approach is not specific towards ReLU and can approximate complex AFs (e.g., SiLU, GeLU, Mish) — satisfying design goal ❶ as described in Section 3.2. Furthermore Equation (4), and (5) comprise of three math operations ADD, MUL, and COMP and the majority of the MPC libraries support these three math operations and thus Compact generated approximations of AF satisfy design goal ❸.

However, generally approximation-based approaches tend to be inaccurate [42]. Thus, maintaining negligible accuracy loss with increasing hidden layers (design goal ❷) and imposing no restriction on training (design goal ❹) at the same time become challenging. We address this challenge by developing the techniques described in the subsequent sections.

```
// Global parameters used by Compact
F_act ← activation function to approximate between [s, e]
P ← N(0,1); E ← E_Mean; s ← −5; e ← 5
η ← plaintext inference accuracy using F_act
ν ← 10^−2 // accuracy loss practitioners can tolerate.

GenAccuracteApprox (θ):
⟨m, k, R⟩ ← θ
Δ ← (e − s)/m
F̂^best_act ← ∅;  δ_lo ← 0;  δ_hi ← MAX_APPROX_ERROR
while δ_lo < δ_hi
    F̂_act ← ∅;  δ_mid ← (δ_lo + δ_high)/2;  α ← s;  β ← α + Δ
    while β < e:
        f ← InterPolate (F_act, k, α, β)
        δ' ← E (P, F_act, f, α, β)
        if δ' > δ_mid/m:
            F̂_act ← F̂_act ⋃{f};  α ← β
        β ← β + Δ
    η' ← compute accuracy using F̂_act on R
    if (η − η')/η ≤ ν ∧ |F̂_act| ≤ m :
        F̂^best_act ← F̂_act;  δ_hi ← δ_mid
    else:  δ_lo ← δ_mid
return F̂^best_act
```

```
FindBestPiecePoly():
θ_cur ← ⟨m_0, k_0, R_0⟩ // Initial solution
F̂^cur_act ← GenAccuracteApprox(θ_cur)
// Solving the constraint optimization problem in Equation (10)
for i ← 1 to i_max // i_max ← 10 for our experiments
    T_i ← χ_0/log(1 + i) // χ_0 ← 0.2 for our experiments
    θ_i ← GenerateNeighbour(θ_cur)
    F̂^i_act ← GenAccuracteApprox(θ_i)
    if F̂^i_act = ∅:  continue // cannot find an F̂_act with (η − η')/η ≤ ν
    r ←$ U_[0,1]
    if exp((Time(F̂^cur_act) − Time(F̂^i_act))/T_i) > r:
        F̂^cur_act ← F̂^i_act;  θ_cur ← θ_i
return F̂^cur_act

GenerateNeighbour(θ):
⟨m, k, R⟩ ← θ
Sample z_1, z_2 ∈ Z according to the probability density function p(z) = 1/3 · 2^−|z|
m' ← m + z_1;  k' ← k + z_2
ℓ' ←$ {128, 84, 64, 32}; γ_2 ←$ {1.5, 2, 2.5, 3, 3.5, 4}
d' ← ⌊ℓ'/γ_2⌋
θ' ← ⟨m', k', R'_⟨ℓ',d'⟩⟩
return θ'
```

Figure 4: (Right-top) FindBestPiecePoly procedure to find an MPC-friendly approximation $\widehat{F}_{\text{act}}$ of the complex activation function (AF) F_{act} (Section 4.3). The procedure balances the trade-off between inference accuracy loss and performance overhead using an application-specific optimization approach (simulated annealing). It uses two sub-procedures—GenerateNeighbour to generate a random neighbor θ' from a given θ (shown Right-bottom) and GenAccuracteApprox to approximate the region $[s, e]$ accurately using a set of at most m polynomials (shown Left) with degree $\leq$ k (Section 4.2). Notations are explained briefly in Table 2.

4.2 Generating Accurate Approximations

To approximate F_{act} for a given region $[s, e]$ using m piece-wise polynomials with degree at most k and has negligible accuracy loss, we use an opportunistic approach GenAccuracteApprox as shown in Figure 4. We use an interpolation technique similar to the one proposed in NFGen [21]. However we use dynamic approximation which makes Compact computationally more efficient than NFGen as the number of hidden layers increases (experimentally illustrated in Section 5.4).

4.2.1 Computing $P(x)$. We aim to design approximate polynomials that are close to accurate on likely values of x, meaning higher probability according to $\mathsf{P}(x)$, while may have higher error on values of x, which are less likely. A challenge, however, is how to estimate the distribution P. Interestingly, in DNN models, the inputs to an AF are first batch normalized (BN) using Equation (2), to help the network converge faster during training (discussed earlier in Section 2.1). Therefore, the set of values the AF is computed on is distributed (approximately) as a normal distribution with zero mean and unit standard deviation. The approximating piecewise polynomial, therefore, should ensure low error on highly likely inputs, whereas on low probable inputs, it may allow making a higher error.

Our key insight is that $\mathsf{P}(x)$ can guide us to focus on approximating those regions more accurately where $\mathsf{P}(x)$ is high. We estimate $\mathsf{P}(x)$ using a standard normal distribution $\mathcal{N}(0, 1)$, and use a customized function to compute the approximation error that takes into account $\mathsf{P}(x)$, as we describe next. We remark on one caveat of this design choice: Compact becomes reliant on BN as we discuss further in Section 6.

4.2.2 Designing $\mathcal{E}$. To incorporate $\mathsf{P}(x)$ to the approximation procedure, we customize an approximation error which we refer to as *weighted mean approximation error* denoted by $\mathcal{E}_{\text{Mean}}$.

$$\mathcal{E}_{\text{Mean}}(\mathsf{P}, F_{\text{act}}, f, \alpha, \beta) = \frac{1}{(\beta - \alpha)} \int_{\alpha}^{\beta} \mathsf{P}(x) \cdot |F_{\text{act}}(x) - f(x)| dx \quad (6)$$

As shown in Equation (6), in addition to considering how accurately $f(x)$ estimates $F_{\text{act}}(x)$ for a given region between α and β, $\mathcal{E}_{\text{Mean}}$ also takes $\mathsf{P}(x)$ into account. NFGen [21] uses *max approximation error*, which we denote by $\mathcal{E}_{\text{Max}}$ as a way to design $\mathcal{E}$.

$$\mathcal{E}_{\text{Max}}(F_{\text{act}}, f, \alpha, \beta) = \max_{x \in [\alpha, \beta]} |F_{\text{act}}(x) - f(x)|$$

We choose to use $\mathcal{E}_{\text{Mean}}$ over $\mathcal{E}_{\text{Max}}$ as it is easy to guide the approximation process via $\mathsf{P}(x)$ using $\mathcal{E}_{\text{Mean}}$.

4.2.3 Selecting a threshold δ for approximation error $\mathcal{E}_{\text{Mean}}$. A straightforward ad hoc way to ensure the accuracy of the approximation is to set a fixed approximation error threshold (δ) and consider an approximation accurate if approximation error calculated via $\mathcal{E}$ is $\leq \delta$. NFGen follows this ad hoc approach and sets $\delta = 10^{-3}$. Via empirical experimentation, they observed that if $\mathcal{E}_{\text{Max}} \leq 10^{-3}$, then the generated approximation, when used in logistic regression and χ^2 testing, does not degrade accuracy without adding much performance overhead.

We refrain from setting a fixed δ for Compact as the appropriate δ may vary from one DNN model or dataset to another. Also, Compact should systematically find the appropriate δ, relieving the practitioners of the additional burden of finding an appropriate δ on their own. Thus, Compact discovers an appropriate δ by performing a binary search over δ and finding the highest δ such that the approximation corresponding to δ incurs a negligible inference accuracy loss. This is sound due to the monotonic relationship between approximation error and inference accuracy. Lastly, one challenge of this approach is checking if the inference accuracy loss is small at each step of the binary search. We describe a solution to this challenge next.

4.2.4 Measuring accuracy loss. It is difficult to analytically find the optimal δ that minimizes inference accuracy loss while having reduced performance overhead. We attempt to handle this challenge empirically by relying on well-known *closed-world* assumption used in machine learning (i.e., for each testing class enough representative examples are available in the training dataset). More specifically, we replace the original F_{act} with the generated MPC-friendly approximation $\widehat{F}_{\text{act}}$ and calculate the inference accuracy over the training dataset. We call this inference accuracy η' and compare it with the plaintext inference accuracy η which uses the original F_{act} over the same training dataset. If $(\eta - \eta')/\eta \leq \nu$, we consider $\widehat{F}_{\text{act}}$ to be accurate enough, where ν is a small value representing the accuracy loss the practitioner can tolerate for switching to secure inference from plaintext inference.

4.2.5 Designing $\widehat{F}^{\text{crd}}_{\text{act}}$. We also added another DNN model-specific optional optimization. Instead of approximating the original $F_{\text{act}}(x)$, we manually introduce a crude MPC-friendly approximation of $F_{\text{act}}(x)$, which we call $\widehat{F}^{\text{crd}}_{\text{act}}$.[2] Then, we use GenAccuracteApprox procedure to approximate $\left(F_{\text{act}}(x) - \widehat{F}^{\text{crd}}_{\text{act}}(x)\right)$. The final approximation of an AF would be $\widehat{F}^{\text{crd}}_{\text{act}}(x) + \widehat{F}_{\text{act}}(x)$. Note that $\widehat{F}^{\text{crd}}_{\text{act}}$ is designed to be simple and linear, making it easy to use with standard MPC libraries. We find this approach significantly improves the approximation procedure

For SiLU AF, since $\text{SiLU}(x) = x \cdot \text{sigmoid}(x)$, we can simply borrow the structure of the MPC-friendly approximation for $\text{sigmoid}(x) \approx \max(0, \min(x{+}0.5, 1))$ from [59]. We tweak it slightly to be more precise and multiply it by x to get $\widehat{F}^{\text{crd}}_{\text{silu}}$ as shown in Equation (7).

$$\widehat{F}^{\text{crd}}_{\text{silu}}(x) = x \cdot \max\left(0, \min(6x + 0.5, 1)\right) \tag{7}$$

For GeLU AF, since $\text{GeLU}(x) \approx x \cdot \text{sigmoid}(1.702x)$, similarly we can write crude MPC-friendly approximation of GeLU AF by leveraging the same structure of MPC approximation for sigmoid as shown in Equation (8).

$$\widehat{F}^{\text{crd}}_{\text{GeLU}}(x) = x \cdot \max\left(0, \min(10x, 0.5)\right) \tag{8}$$

Since Mish cannot be expressed easily in terms of sigmoid, we denote crude MPC friendly approximation of it by ReLU as shown in Equation (9).

$$\widehat{F}^{\text{crd}}_{\text{Mish}}(x) = \max(0, x) \tag{9}$$

[2]For simplicity this is not shown in Figure 4 GenAccuracteApprox.

4.2.6 Performing interpolation. We interpolate $f(x)$ between range $[\alpha, \beta]$ by a k-degree polynomial f (Equation (5)) using the InterPolate procedure. To find the best performing $f(x)$, similar to NFGen, we adopt Chebyshev interpolation [53] over other alternatives, such as cubic spline or uniform polynomial. This is due to an established fact in the area of function approximation theory [55] that Chebyshev polynomial interpolation generally has superior performance to cubic spline or uniform polynomials interpolation when $f(x)$ involves non-linear operations such as e^{-x}, ln, tanh, as it is the case with complex AFs shown in Figure 1.

GenAccuracteApprox procedure. Now we can piece together the above-mentioned techniques and describe the procedure to approximate F_{act} within region $[s, e]$ using a number of piece-wise polynomials in detail (as shown in GenAccuracteApprox Figure 4). First, we set a step size Δ, $\alpha \leftarrow s$, and $\beta \leftarrow \alpha + \Delta$. Then at each step, we increase the pointer β by Δ. Before increasing β, we check if the adjusted approximation error δ' in the region $[\alpha, \beta]$ is more than the expected approximation error δ/m.

If this is the case, we approximate the region $[\alpha, \beta]$ using a polynomial f using the Chebyshev interpolation algorithm, add that polynomial piece to $\widehat{F}_{\text{act}} \leftarrow \widehat{F}_{\text{act}} \bigcup \{f\}$, and update α to β. Next, we update β by Δ, and again perform the above-mentioned check until we have approximated the whole region $[s, e]$.

4.3 Finding Efficient Approximation

Now that we can generate MPC-friendly approximations $\widehat{F}_{\text{act}}$ using GenAccuracteApprox procedure that have negligible accuracy loss for a given $\langle \text{m}, \text{k}, \mathcal{R} \rangle$, one can search over all possible values of $\langle \text{m}, \text{k}, \mathcal{R} \rangle$ and select the $\widehat{F}_{\text{act}}$ that is computationally more efficient. We use θ to represent $\langle \text{m}, \text{k}, \mathcal{R} \rangle$. We also use $\text{Time}(\widehat{F}'_{\text{act}})$ to represent the average time it takes to complete secure inference with the approximation $\widehat{F}'_{\text{act}}$ generated using θ' in the approximation process. The accuracy loss can be presented by $\text{AccLoss}(\widehat{F}'_{\text{act}}, F_{\text{act}}) = (\eta - \eta')/\eta$; where η is the accuracy when we use the complex AF F_{act} as it is (i.e., plaintext accuracy), and η' is the accuracy when we replace the complex AF with its MPC-friendly approximation $\widehat{F}'_{\text{act}}$ (i.e., secure inference accuracy).

Unfortunately, because of performing the binary search to find the appropriate δ, GenAccuracteApprox becomes time-consuming. This is because determining if the accuracy loss is negligible at each step of binary search with reasonable confidence requires performing inference over the large training dataset (as explained in Section 4.2.4), and it makes exhaustively iterating over all possible values of θ from the search space Θ infeasible.

In this work, we treat this problem of finding optimal θ used for the approximation to balance performance overhead and accuracy loss, as a constraint optimization problem (COP). Roughly, this means, we find a $\theta' \in \Theta$ that minimizes the average inference time under the constraint that the accuracy loss is less than a specified threshold ν.

Concretely, for a given F_{act}, we want to solve the following optimization problem.

$$\theta \leftarrow \underset{\theta' \in \Theta}{\text{argmin}}\ \text{Time}(\widehat{F}'_{\text{act}}) \text{ s.t } \text{AccLoss}(\widehat{F}'_{\text{act}}, F_{\text{act}}) \leq \nu \tag{10}$$

Here ν is the specified maximum accuracy loss threshold we can tolerate, and $\widehat{F}'_{\text{act}}$ is the MPC-friendly approximation of F_{act} generated by Compact using θ'. Now to solve this COP problem in Equation (10), we devise a search technique based on simulated annealing (SA) [40]. SA is a general framework to tackle COP. Briefly, SA starts with an initial solution, generates new neighboring solutions relative to the current solution, and probabilistically decides between moving to the new solution or staying with the current solution for fixed number of iterations. One advantage of sketching SA-based searching for optimal solution θ is that SA is gradient-free — suiting our needs — overcoming the difficulty to underpin an analytical formula of $\nabla_{\theta=\langle \mathsf{m},\mathsf{k},\mathcal{R}\rangle}\mathsf{GenAccuracteApprox}(\cdot)$. That being said, other gradient-free searching techniques may also work as well [20], and we detail some additional discussions about this in Appendix B.

One important characteristic of SA—we need to model for this case—is how to avoid being trapped in a local suboptimal solution. To this extent, we follow suggestions from prior work [15, 37], and *probabilistically* move towards a new solution θ_i even if θ_i is computationally less efficient approximation than the current best solution (θ_{cur}).

More precisely, if at i-th iteration, we denote the MPC-friendly approximation from θ_i as $\widehat{F}^i_{\text{act}}$, then we always update our current best solution θ_{cur} to θ_i if $\widehat{F}^i_{\text{act}}$ is computationally more efficient than $\widehat{F}^{\text{cur}}_{\text{act}}$ (i.e., $\mathsf{Time}(\widehat{F}^{\text{cur}}_{\text{act}}) > \mathsf{Time}(\widehat{F}^i_{\text{act}})$). Otherwise, we update θ_{cur} to θ_i with a certain acceptance probability. This probability depends on two factors. First, the temperature at i-th iteration called T_i — which is initially high, meaning we have a high tendency to accept a solution computationally less efficient, but after a few more iterations T_i decreases and so does our tendency to accept a computationally less efficient solution. Second, the amount of computation less efficient $\widehat{F}^i_{\text{act}}$ is compared to $\widehat{F}^{\text{cur}}_{\text{act}}$. In other words, we accept θ_i when $\exp(\psi/T_i) > r$ is true. Here ψ represents the computational efficiency of $\widehat{F}^i_{\text{act}}$ over the current approximation $\widehat{F}^{\text{cur}}_{\text{act}}$ expressed as $\psi \leftarrow \mathsf{Time}(\widehat{F}^{\text{cur}}_{\text{act}}) - \mathsf{Time}(\widehat{F}^i_{\text{act}})$, and r is a randomly chosen number given by $r \leftarrow\!\!\$\ U_{[0,1]}$.

We have to design two more parameters carefully. One is the neighborhood generation heuristic for θ, and the other is setting a cooling schedule for the temperature T_i. Without careful handling of these two parameters SA may lead to undesired approximations [6].

Neighbour generation heuristic. At iteration i, we generate a new neighbor $\theta_i = \langle \mathsf{m}', \mathsf{k}', \mathcal{R}'\rangle$ from $\theta = \langle \mathsf{m}, \mathsf{k}, \mathcal{R}\rangle$ in the following way: for m', k' we randomly sample two integer numbers $z_1, z_2 \in \mathbb{Z}$ from a probability distribution having density function $p(x = z) = (1/3) \cdot 2^{-|z|}$ such that and set $\mathsf{m}' \leftarrow \mathsf{m} + z_1$ and $\mathsf{k}' \leftarrow \mathsf{k} + z_2$. This means that the chances of moving further away from the current value m and k decreases exponentially.

To specify a $\mathcal{R}$, we need two numbers: *i)* the size of the ring used in MPC library (denoted by ℓ), and *ii)* the number of last bits to represent the fractional parts (denoted by d). Typically, MPC libraries use $\mathcal{R}$ sizes of $\{128, 84, 64, 32\}$. We randomly sample a ring size from these for ℓ, and regarding values of d we set it to $d \leftarrow \lfloor \ell/\gamma_2 \rfloor$ where γ_2 is randomly sample from $\gamma_2 \in_R \{1.5, 2, 2.5, 3, 3.5, 4\}$.

Cooling schedule. As for the cooling schedule, we adopt the classical logarithmic series $T_i \leftarrow \chi_0/\log(i+1)$ at i-th iteration following Hajek et al. [28]. This choice ensures that initially, T_i would be high, thereby increasing the chances of accepting a computationally less efficient approximation during the early iterations. But as the number of iterations increases, T_i progressively decreases, lowering this chance. We simply set $\chi_0 = 0.2$ for all of our experiments, yielding $T_1 \approx 0.67$ and $T_{10} \approx 0.2$.

We show the pseudocode for finding computationally efficient approximation FindBestPiecePoly and the procedure for generating neighbors at each iteration GenerateNeighbour in Figure 4.

5 EXPERIMENTAL EVALUATION

We conduct experiments to address the following questions:

(1) *Model Accuracy* (Section 5.3): What is the impact on model inference accuracy of using MPC-friendly activation functions $\widehat{F}_{\text{act}}(x)$ generated using our scheme Compact and other existing approaches [21, 48, 51, 67]?

(2) *Inference Time* (Section 5.4): What is the inference time overhead of Compact compared to NFGen [21] as the number of hidden layers increases?

5.1 Implementation Details

Our Scheme. We implement our scheme using Python 3.8 in about 800 lines of code. We approximate the region between $x \in [-5, 5]$ for all activation functions (AFs) as beyond that region, they can be easily approximated using simple polynomials. Also, we use SymPy [54] library for the majority of mathematical operations, including calculating the approximation error between a given region of a polynomial using Equation (6) and performing Chebyshev interpolation as mentioned in Section 4.2.6. Our source code is publicly available [39].

Our scheme requires testing if the generated approximation has negligible accuracy loss by checking $(\eta - \eta')/\eta \leq \nu$ (as described in Section 4.2.4). We also configure FindBestPiecePoly with ten iterations ($i_{max} = 10$) to find a computationally efficient approximation and set $\chi_0 = 0.2$. For the initial solution $\theta_0 = \langle \mathsf{m}_0, \mathsf{k}_0, \mathcal{R}_0\rangle$, we set $\mathsf{m}_0 = 10^4$ and $\mathsf{k}_0 = 10$ (default parameters taken from NFGen [21]). For $\mathcal{R}_{\langle \ell_0, d_0\rangle}$, we use $\langle \ell_0, d_0\rangle = \langle 128, 64\rangle$ — a popular choice of ring size by many MPC libraries. With this configuration, FindBestPiecePoly took less than 25 minutes on commodity hardware to finish the four tasks and three complex AFs we detail in Section 5.2. Table 6 shows the appropriate $\mathsf{m}, \mathsf{k}, \mathcal{R}$ we find via FindBestPiecePoly for all tasks and complex AFs.

Other Approaches. We consider four state-of-the-art approaches for comparison: NFGen [21], MiniONN [51], MPCFormer [48] and SIRNN [67]. Additionally, we consider a rudimentary base approach: replacing the complex AF with a popular MPC-friendly AF ReLU. We consider this approach as ReLU is relatively MPC-friendly because it can be computed using only two piece-wise polynomials.

For NFGen, we add a wrapper class to the author's open-source implementation to measure the inference accuracy and computational overhead for the four tasks. Besides that, we keep their implementation unchanged — using $\mathcal{E}_{\mathsf{Max}}$ (Equation (4.2.2)) to measure the approximation error, setting $\delta = 10^{-3}$, $\mathsf{k} = 10$, and $\mathsf{m} = 10^4$. Liu et al. [51] describe an approach called MiniONN for generating MPC-friendly approximations of sigmoid AF. Since there

is no publicly available implementation of MiniONN, we implement it ourselves based on the description given in [51] and extend the approach to generate MPC-friendly versions of complex AF $F_{\text{act}} \in \{\text{SiLU}, \text{Mish}, \text{GeLU}\}$. Further details of MiniONN, and our extension are discussed in Appendix C.

MPCFormer [48] approximates GeLU using an MPC-friendly polynomial given by $\text{GeLU}(x) = 0.125x^2 + 0.25x + 0.5$. This approximation was motivated by the need to perform secure inference for transformer-based DNN models where GeLU activation is used extensively. Since Li et al. [48] did not provide any recipes that could be generalized directly to other AF, we only compare the accuracy and computational overhead for GeLU.

Lastly, Rathee et al. [67] present a library called SIRNN that computes complex mathematical operations (e.g., $e^x, \ln(x), \frac{1}{x}$) securely using a combination of lookup tables and numerical methods (e.g., Goldschmidt's iterations). Thus, complex AFs can be computed sequentially by performing the aforementioned operations and combining the intermediate results using ADD, MUL, COMP operators to evaluate F_{act}. Recently, Hao et al. [29] extended their approach to computing GeLU AF efficiently by reducing one network call. Nevertheless, this work uses the open source implementation of SIRNN [60].

5.2 Experimental Setup

To demonstrate that inference accuracy and performance overhead is negligible for secure inference using our scheme, we consider four state-of-the-art image classification tasks as shown in Table 3 and three complex activation functions (AF) $F_{\text{act}} \in \{\text{SiLU}, \text{GeLU}, \text{Mish}\}$. We train the four models corresponding to each complex AF for each task. While training these models, we preserve the widely use parameters as proposed in the literature for all models (e.g., the overall architecture of the model, # of epochs, learning rate, optimizer, etc.) — including a batch normalization layer before inputs are being fed to complex activation functions of each hidden layer as illustrated in Figure 2.

Below, we provide brief details about these four classification tasks, and further details are in Appendix D.

Four classification tasks. For the first task, we consider a simple classification task of MNIST dataset [46] using a three-layer deep fully connected network (FCN) with one input, output, and hidden layer. MNIST dataset contains 70 thousand 28×28 handwritten digits grey images, and the three-layer deep FCN achieves close to 0.99 training accuracy for the three complex AFs. We refer to this task as DigitRecognition in the paper. Next, we move towards a more complex classification task of CIFAR-10 dataset [44] — which we refer to as CIFAR10Classification.

CIFAR-10 consists of 60 thousand 32×32 color images with six thousand images per 10 classes. For performing classification on this dataset, we use a a convolutional neural network (ConvNet) [64] with five hidden layers and train it over the 50 thousand training images of CIFAR-10 dataset using three different complex AFs. For the third task, we consider performing classification on ImageNet-1K dataset which has been one of the challenging benchmark datasets in image classification [16]. We refer to this task as ImageNet1KClassification in this paper.

The ImageNet-1K dataset contains around one million annotated images with 50 thousand validation images and 100 thousand test images. We use a deep residual neural network (ResNet9) [30] model having eight hidden layers over the training images for 50 epochs for three complex activation functions and achieved a validation accuracy of around 0.74. Lastly, we perform experiments to detect spoofed images in CelebA-Spoof [83] dataset. We refer to this task as SpoofFaceDetection in this paper.

CelebA-Spoof is a large-scale face anti-spoofing dataset used to train anti-spoofing DNN models. CelebA-Spoof contains 625 thousand facial images from around 10 thousand subjects with each image having 43 attributes; 40 of them correspond to indicating facial components of real images and three of them correspond to attributes of spoofed facial images. For training, we perform an 80%-20% split of the CelebA-Spoof dataset and adopted the EfficientNetB0 [74] model, which is the state-of-the-art top-performing anti-spoofing detection model and winner of the CVPR challenge of detecting spoofed face images [50]. The EfficientNetB0 model consists of 17 hidden layers, and after training the model for 25 epochs, it achieved a training accuracy of 0.98.

Machine specification. We train the models on a Linux machine with an Intel Core i9 processor having 128 GB RAM and Nvidia GTX 1080 GPU. The training split of each dataset is used for training the models. After the training is completed, we save these models. We assume $\mathcal{S}_{\text{owner}}$ holds these saved models and the weights of these models are $\mathbf{W}$ which $\mathcal{S}_{\text{owner}}$ does not want to reveal to $\mathcal{C}$ while performing secure inference. We simulate the $\mathcal{C}$'s input $\mathbf{x}$ using the testing split of the corresponding datasets for each task.

5.3 Model Accuracy

We first measure the inference accuracy of the trained models over the testing split of the dataset by using the (non-MPC-friendly) complex activation functions as it is and refer to it as *plaintext accuracy* (η). Then, we replace the complex activation function with its MPC-friendly approximation generated by different approaches and measure its inference accuracy (η'). Thus, $\nu = (\eta - \eta')/\eta$ gives the inference accuracy loss introduced by MPC-friendly approximations. Table 3 shows the inference accuracy loss (in percentage) for Compact generated MPC-friendly approximation and other state-of-the-art approaches generated MPC-friendly approximation [21, 29, 51] — for each task across the three complex AFs SiLU, GeLU, Mish.

Now we discuss the inference accuracy loss for different approaches, and throughout the discussion, we conservatively consider accuracy loss negligible if $\nu < 10^{-2}$.

ReLU based approach. We observe that although for the first DigitRecognition task, the inference accuracy loss is within 1.54%-2.68% for the last three tasks accuracy loss is higher — at least 45.66% — making this approach unsatisfactory.

SIRNN [67]. For SIRNN, we observe that for DigitRecognition task, we observe less significant accuracy loss (0.95%–2.37%). Furthermore for SpoofFaceDetection the accuracy does not degrade too much — by 0.48%–1.78%. However, for CIFAR10Classification and ImageNet1KClassification task the accuracy degradation is higher — suffering from an accuracy loss of 2.58%–16.31%.

Task Name	Model†	Dataset	F_{act}	% Plaintext accuracy	% Accuracy loss§ ReLU	NFGen	MiniONN	MPCFormer	SIRNN	Compact
DigitRecognition	FCN	MNIST	SiLU	98.73	2.31	0.43	20.88	*n/a*	2.37	0.17
			GeLU	98.45	1.54	0.23	42.31	0.18	1.32	0.97
			Mish	99.07	2.68	0.19	30.41	*n/a*	0.95	0.06
CIFAR10Classification	ConvNet	CIFAR-10	SiLU	86.53	49.80	0.51	18.50	*n/a*	2.58	0.49
			GeLU	87.11	45.66	0.64	30.04	7.07	4.01	0.25
			Mish	89.30	57.07	0.27	57.07	*n/a*	13.64	0.11
ImageNet1KClassification	ResNet9	ImageNet-1K	SiLU	72.89	98.39	1.36	27.12	*n/a*	10.59	0.91
			GeLU	75.43	77.66	0.05	36.21	9.43	6.68	0.03
			Mish	75.78	98.97	0.61	39.89	*n/a*	16.31	0.55
SpoofFaceDetection	EfficientNetB0	CelebA-Spoof	SiLU	90.87	71.72	0.14	4.27	*n/a*	1.75	0.08
			GeLU	92.19	75.94	0.20	9.75	0.09	0.48	0.77
			Mish	92.23	77.71	0.53	1.32	*n/a*	1.78	0.66

n/a = MPCFormer does not propose MPC friendly approximation for SiLU, Mish.

§Accuracy loss is reported by comparing the inference accuracy η and η' obtained using AF F_{act} and $\widehat{F}_{act}$, respectively. Accuracy loss = $(\eta - \eta')/\eta$, and reported in percentage (%). Accuracy losses of $< 10^{-2}$ or $< 1\%$ are highlighted in gray.

†For all models, batch normalization is used before each activation layer.

Table 3: Inference accuracy of MPC-friendly approximation of three complex activation functions (AF) for four different tasks using state-of-the-art approaches. Except for NFGen and our approach other DNN-specific approaches show a significant drop in inference accuracy if we use their generated MPC-friendly version of complex AF. We compare the performance overhead of our approach with NFGen in Section 5.4 and show results in Table 4.

We hypothesize such accuracy degradation is primarily due to two reasons: 1) intermediate steps overflowing in the fixed point representation, and 2) errors introduced in one layer propagating to subsequent layers and accumulating in the process. This further motivates the need to take a piece-wise polynomial approximation-based approach for designing MPC-friendly approximation of complex AF when state-of-the-art DNN models are used, confirming findings from prior work [21].

MPCFormer [48] For MPCFormer, we observe a negligible accuracy loss for DigitRecognition and SpoofFaceDetection tasks of 0.18% and 0.09% respectively. However, similar to SIRNN, it exhibits a non-negligible accuracy loss of 9.4% and 7.07% for CIFAR10Classification and ImageNet1KClassification task respectively. We suspect this is because GeLU activation approximation by MPCFormer relies on *knowledge distillation* (KD) [33] – which is essentially fine-tuning the sequence-to-sequence-based pre-trained model for efficiency. In absence of KD, a simple plug-and-play replacement of polynomial approximation of GeLU activation proposed by MPCFormer does not work well.

MiniONN [51]. For MiniONN, we observe that the inference accuracy loss becomes significant when we use their recipe to generate a friendly approximation of complex AFs SiLU, GeLU, Mish. The accuracy loss becomes catastrophically high, especially for ImageNet1KClassification, (27.12%–39.89%). This shows that although the recipe proposed by MiniONN does not show accuracy degradation for sigmoid AF for simplistic logistic regression models, there is a generalization gap when such recipes are used for DNN models trained on diverse datasets involving complex AFs.

Compact and NFGen [21] We observe that for all tasks, in general, Compact and NFGen generated MPC-friendly approximations have negligible accuracy loss of $< 1\%$. For one instance, though, ImageNet1KClassification task involving SiLU AF, NFGen has an accuracy loss of 1.36% — marginally higher than the aforementioned threshold. When comparing the two approaches, generally, Compact generated approximation has lower accuracy loss, except for two instances showing a slight deviation. The first one is DigitRecognition task involving GeLU (0.37% vs 0.23%) and SpoofFaceDetection task involving Mish AF (0.66% vs 0.53%).

Results summary. We conclude from these experiments that NFGen and Compact are resistant to significant accuracy loss — when we use their generated MPC-friendly approximation instead of the original complex AF — compared to other approaches we consider. Keeping that in mind, we can now investigate the next important aspect of secure inference, measuring performance overhead. We narrow down our experiments to NFGen and Compact — excluding the other approaches — as their accuracy loss is significantly high.

5.4 Inference Time

We benchmark the inference time of NFGen and Compact to measure the performance overhead. While benchmarking, we instantiate each party in the protocol by machines running on commodity-type hardware — having an Intel Core i7 processor with 64 GB RAM and connected over a 252 Mbits/sec network link. We use the average inference time for a single image calculated over the testing split of the datasets and include both computational and communication costs while reporting the results.

We consider two state-of-the-art MPC libraries [58, 68] designed for secure inference — one for a 2PC scenario and the other for a 3PC scenario (as described earlier in Section 3).

3PC results. First, for the 3PC scenario, we consider ABY3 [58] that uses replicated secret sharing (SS) based secure inference protocol. Table 4 compares the performance overhead of Compact and

Task Name	# HLs†	F_{act}	NFGen	Ours	**Speedup**
DigitRecognition	1	SiLU	40	43	0.93×
		GeLU	35	32	1.09×
		Mish	52	49	1.06×
CIFAR10Classification	5	SiLU	114	58	1.96×
		GeLU	194	94	2.05×
		Mish	117	62	1.89×
ImageNet1KClassification	8	SiLU	359	102	3.52×
		GeLU	446	106	4.17×
		Mish	473	104	4.52×
SpoofFaceDetection	17	SiLU	204	47	4.34×
		GeLU	221	45	4.91×
		Mish	195	41	4.75×

† # HLs = Number of hidden layers.

Table 4: Comparison of inference time (ms) of three activation functions (F_{act}) over four different classification tasks for $n = 3$ computing servers using ABY3 library.

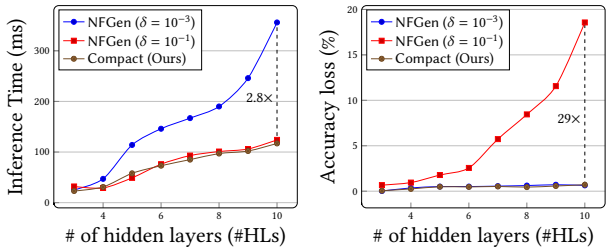


Figure 5: Comparison of inference time (left), and accuracy loss (right) of NFGen with Compact. For both lower is better. We experiment with two approximation error thresholds for NFGen: i) $\delta = 10^{-1}$ (a crude one we set) and ii) $\delta = 10^{-3}$ (used in [21]). NFGen ($\delta = 10^{-1}$) achieves a lower inference time but has significant accuracy loss—while NFGen ($\delta = 10^{-3}$) shows the opposite characteristic when we compare them with Compact as #HLs increases. Compact performs well on both accounts.

NFGen for the 3PC setting using ABY3 library. We observe that Compact outperforms NFGen 2×-5× for the last three classification tasks involving a high number of layers. However, Compact's performance efficacy for the first task DigitRecognition is comparable to NFGen—exhibiting similar inference time. We hypothesize this is because the number of hidden layers is only one for the DNN model used in this first task. In contrast, the number of hidden layers for the other classification tasks is 5, 8, and 17 respectively. Because of this, there is a higher chance of the approximation errors introduced in one hidden layer propagating to the next hidden layers. Our DNN-specific techniques discussed in Section 4 can effectively curb out this approximation error from propagating to the hidden next layers without sacrificing much performance overhead compared to NFGen. Thus, Compact's superior performance becomes more pronounced as number of hidden layers becomes high.

2PC results. For the 2PC scenario, we consider CryptFlow2 [68] another state-of-the-art library for secure inference based on a novel protocol for *millionaries' problem* [82] and division over fixed-point arithmetic. We experiment with the oblivious transfer (OT) based construction of CryptoFlow2 but believe performance results would also be similar for homomorphic encryption (HE) based construction. We observe a performance efficiency trend of Compact with NFGen similar to the 3PC scenario. We present detailed results of those experiments in Appendix E.

Ablation study. From the above experiments, it is clear that Compact is 2×—5× computationally more efficient than NFGen while maintaining a negligible accuracy loss. We perform an ablation study to investigate further what makes Compact more efficient than NFGen. For this study, we train 10 convolutional neural networks on the CIFAR-100 dataset having 1 to 10 hidden layers. We consider two approximation error thresholds δ for NFGen: $\delta = 10^{-3}$ as used by Fan et al. [21], and $\delta = 10^{-1}$ set by us. Figure 5 shows the percentage accuracy loss and inference time for 3PC scenario and SiLU activation function.

We observe that, on the one hand, although NFGen with a crude approximation error threshold $\delta = 10^{-1}$ generates an approximation of SiLU that renders inference time closer to Compact with increasing number of hidden layers, it suffers from significant accuracy loss. On the other hand, NFGen with $\delta = 10^{-3}$ has negligible accuracy loss similar to Compact. But it has a high inference time with increasing hidden layers.

We believe this is due to two reasons. First, Compact automatically finds the optimal $\theta = \langle m, k, \mathcal{R} \rangle$ via binary search over δ coupled with a simulated annealing-based heuristic. Using a fixed δ as NFGen fails to achieve this. Second, we incorporate batch normalization into approximation error calculation $\mathcal{E}_{\text{Mean}}$ (Equation (4.2.2)) to have a better approximation to polynomial pieces near the zero regions. This is different from the approximation error calculated by NFGen, which uses $\mathcal{E}_{\text{Max}}$. As the number of complex AFs increases from with the number of layers, Compact's small efficiency for a single complex AF over NFGen becomes pronounced.

6 CONCLUSION AND FUTURE WORK

In this work, we present Compact that enables fast secure inference for DNN models that use complex activation functions (AFs). We experimentally validate that Compact have better performance and inference accuracy on the three most popular complex AFs (i.e., SiLU, GeLU, Mish) than the state-of-the-art approaches when the number of hidden layers is more than one. Thus, we believe Compact can accelerate easy adoption of secure inference even when DNN models have a number of hidden layers, trained over complex AFs to generate robust, noise-resistant, better-performing DNN models. Deploying Compact is straightforward as it is compatible with standard MPC libraries, and obviates the need to retrain/fine-tune DNN models further after replacing the complex AF with ReLU AF to make it compatible with secure inference protocols specific to ReLU. Furthermore, one can easily use our approach to approximate other less widely used complex AFs as well (e.g., tanh, Smish, etc.). We point few promising research directions that we hope future work will investigate.

Accelerating secure inference time using GPU. In the plaintext setting, impressive ML inference time has been achieved by harnessing GPUs, which support highly parallelizable workloads.

This boost in inference speed can also be extended to secure setting by running MPC operations inside GPUs. Indeed, recent works have shown how to achieve significant speedup in machine learning training and inference by making MPC operations GPU-friendly [75, 80]. We did not use GPUs while benchmarking secure inference time. Thus, future work can look at porting these GPU-friendly techniques to two state-of-the-art secure inference protocols ABY3 and CryptFlow2, we use for benchmarking and improve the inference time reported in Table 4.

Dependence on batch normalization. As batch normalization (BN) is employed before AFs by the majority of state-of-the-art DNN models used in computer vision, Compact leverages the phenomenon that BN shifts the input distribution to have zero mean and unit variance. We believe other normalization approach, such as layer normalization – typically used in natural language processing – can also be leveraged to design an approach similar to ours in the future.

Robustness of secure inference and training. For safety-critical, and privacy-sensitive applications, understanding the robustness of the model against attacks like training data poisoning, model inversion, adversarial examples, and membership inference attacks is crucial before deploying them in practice. To the best of our knowledge, such attacks are not considered by most prior work [63] on secure cryptographic training and inference. Only recently secure training and inference have demonstrated performance levels suitable for practical deployment. Therefore, it is now important to evaluate the robustness of existing secure training or inference protocols against such attacks, before deploying these applications in practice.

Membership inference attacks. Compact uses the training data (or a holdout data) to find an approximated AF that does not degrade accuracy and performance overhead, and one may assume there is a chance that it could affect membership inference attacks for outliers. In fact, any work introducing a cryptography-friendly approximation of AF, that relies on training or holdout data, to balance accuracy and performance, could be susceptible. For example, Delphi [56] uses the training data to run a planner that replaces certain ReLU AF with cryptography-friendly square function, and SecureML [59] replaces ReLU AF with a new cryptography-friendly approximation. We discuss if using cryptography-friendly approximation of activation functions for secure inference could impact membership inference attacks briefly in Appendix F.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. We also thank Jihye Choi and Maximilian Zinkus for their feedback on the initial draft of this paper. This work was partially completed when the first and the last authors were at Visa Research. This research is supported in part by the University of Wisconsin—Madison Office of the Vice Chancellor for Research and Graduate Education and NSF award #2339679, and US Department of Commerce award #70NANB21H043.

REFERENCES

[1] 2022. Machine Learning as a Service: What It Is, When to Use It and What Are the Best Tools Out There. https://neptune.ai/blog/machine-learning-as-a-service-what-it-is-when-to-use-it-and-what-are-the-best-tools-out-there.
[2] Toluwani Aremu and Karthik Nandakumar. 2023. PolyKervNets: Activation-free Neural Networks For Efficient Private Inference. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. 593–604. https://doi.org/10.1109/SaTML54575.2023.00045
[3] Amazon AWS. 2023. Image Classification - MXNet. https://docs.aws.amazon.com/sagemaker/latest/dg/image-classification.html.
[4] Shayan Aziznejad and Michael Unser. 2019. Deep spline networks with control of Lipschitz regularity. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3242–3246.
[5] Amos Beimel. 2011. Secret-sharing schemes: A survey. In *International conference on coding and cryptology*. Springer, 11–46.
[6] Walid Ben-Ameur. 2004. Computing the initial temperature of simulated annealing. *Computational optimization and applications* 29 (2004), 369–385.
[7] Pakshal Bohra, Joaquim Campos, Harshit Gupta, Shayan Aziznejad, and Michael Unser. 2020. Learning activation functions in deep (spline) neural networks. *IEEE Open Journal of Signal Processing* 1 (2020), 295–309.
[8] Pakshal Bohra, Dimitris Perdios, Alexis Goujon, Sébastien Emery, and Michael Unser. 2021. 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*.
[9] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. 2022. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1897–1914.
[10] Hervé Chabanne, Amaury De Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-preserving classification on deep neural network. *Cryptology ePrint Archive* (2017).
[11] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. 2022. SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 1361–1378. https://www.usenix.org/conference/usenixsecurity22/presentation/chandran
[12] Google Cloud. 2023. Media Translation: Add real-time audio translation directly to your content and applications. https://cloud.google.com/media-translation.
[13] Google Cloud. 2023. Recommendations AI: Deliver highly personalized product recommendations at scale. https://cloud.google.com/recommendations.
[14] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2020. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020), 355–375.
[15] Kalyanmoy Deb. 2012. *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd.
[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[17] Abdulrahman Diaa, Lucas Fenaux, Thomas Humphries, Marian Dietz, Faezeh Ebrahimianghazani, Bailey Kacsmar, Xinda Li, Nils Lukas, Rasoul Akhavan Mahdavi, Simon Oya, et al. 2023. Fast and Private Inference of Deep Neural Networks by Co-designing Activation Functions. *arXiv preprint arXiv:2306.08538* (2023).
[18] Ashlea Ebeling. 2022. IRS To Drop Facial Scan ID.me Verification For Online Accounts. https://www.forbes.com/sites/ashleaebeling/2022/02/07/irs-to-drop-facial-scan-idme-verification-for-online-accounts/?sh=105801ad7c9b.
[19] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks* 107 (2018), 3–11.
[20] Yavuz Eren, İbrahim B Küçükdemiral, and İlker Üstoğlu. 2017. Introduction to optimization. In *Optimization in renewable energy systems*. Elsevier, 27–74.
[21] Xiaoyu Fan, Kun Chen, Guosai Wang, Mingchun Zhuang, Yi Li, and Wei Xu. 2022. 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* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 995–1008. https://doi.org/10.1145/3548606.3560565
[22] Frederic B Fitch. 1944. 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 (1944), 49–50.
[23] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1322–1333.
[24] Karthik Garimella, Nandan Kumar Jha, and Brandon Reagen. 2021. Sisyphus: A cautionary tale of using low-degree polynomial activations in privacy-preserving deep learning. *arXiv preprint arXiv:2107.12342* (2021).

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

[26] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.

[27] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[28] Bruce Hajek. 1988. Cooling schedules for optimal annealing. *Mathematics of operations research* 13, 2 (1988), 311–329.

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

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

[31] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian Error Linear Units (Gelus). *arXiv preprint arXiv:1606.08415* (2016).

[32] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).

[33] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[34] John J Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* 79, 8 (1982), 2554–2558.

[35] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 809–826. https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

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

[37] Chii-Ruey Hwang. 1988. Simulated annealing: Theory and applications: PJM van Laarhoven and EHL Aarts: D. Reidel, Dordrecht, 1987, 198 pp., ISBN 90-277-2513-6, Dfl. 120.

[38] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, 448–456.

[39] Mazharul Islam. 2024. Compact. https://github.com/islamazhar/compact-public.

[40] David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. 1989. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations research* 37, 6 (1989), 865–892.

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

[42] Mahimna Kelkar, Phi Hung Le, Mariana Raykova, and Karn Seth. 2022. Secure poisson regression. In *31st USENIX Security Symposium (USENIX Security 22)*. 791–808.

[43] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. 2018. Model Extraction Warning in MLaaS Paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference* (San Juan, PR, USA) *(ACSAC '18)*. Association for Computing Machinery, New York, NY, USA, 371–380. https://doi.org/10.1145/3274694.3274740

[44] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report 0. University of Toronto, Toronto, Ontario.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[47] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. 2021. Muse: Secure inference resilient to malicious clients. In *30th USENIX Security Symposium (USENIX Security 21)*. 2201–2218.

[48] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. 2022. MPCFormer: fast, performant and private Transformer inference with MPC. *arXiv preprint arXiv:2211.01452* (2022).

[49] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen. 2019. Preventing gradient attenuation in lipschitz constrained convolutional networks. *Advances in neural information processing systems* 32 (2019).

[50] Ajian Liu, Chenxu Zhao, Zitong Yu, Anyang Su, Xing Liu, Zijian Kong, Jun Wan, Sergio Escalera, Hugo Jair Escalante, Zhen Lei, et al. 2021. 3D High-Fidelity Mask Face Presentation Attack Detection Challenge. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 814–823.

[51] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 619–631.

[52] Ezequiel López-Rubio, Francisco Ortega-Zamorano, Enrique Domínguez, and José Muñoz-Pérez. 2019. Piecewise polynomial activation functions for feedforward neural networks. *Neural Processing Letters* 50 (2019), 121–147.

[53] Nicholas Marshall. 2024. Chebyshev Interpolation. https://web.math.princeton.edu/~nfm2/chebyshev-interpolation.pdf.

[54] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103. https://doi.org/10.7717/peerj-cs.103

[55] Mario J Miranda and Paul R Fackler. 1996. Lecture Notes In Computational Economic Dynamics; Chapter 5 Function Approximation. http://fmwww.bc.edu/ec-p/software/Miranda/chapt5.pdf.

[56] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2505–2522. https://www.usenix.org/conference/usenixsecurity20/presentation/mishra

[57] Diganta Misra. 2019. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681* (2019).

[58] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 35–52. https://doi.org/10.1145/3243734.3243760

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

[60] mpc msri. 2023. EzPC - a language for secure machine learning. https://github.com/mpc-msri/EzPC/tree/master/EzPC.

[61] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Icml*.

[62] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, and Michael Unser. 2022. Approximation of Lipschitz Functions using Deep Spline Neural Networks. *arXiv preprint arXiv:2204.06233* (2022).

[63] Lucien KL Ng and Sherman SM Chow. 2023. SoK: Cryptographic Neural-Network Computation. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 497–514.

[64] Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

[65] George-Liviu Pereteanu, Amir Alansary, and Jonathan Passerat-Palmbach. 2022. Split HE: Fast Secure Inference Combining Split Learning and Homomorphic Encryption. *arXiv preprint arXiv:2202.13351* (2022).

[66] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).

[67] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. SiRnn: A Math Library for Secure RNN Inference. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1003–1020. https://doi.org/10.1109/SP40001.2021.00086

[68] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.

[69] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. {XONN}:{XNOR-based} Oblivious Deep Neural Network Inference. In *28th USENIX Security Symposium (USENIX Security 19)*. 1501–1518.

[70] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia conference on computer and communications security*. 707–721.

[71] Peter Rindal. 2020. The ABY3 Framework for Machine Learning and Database Operations. https://github.com/ladnir/aby3.

[72] Bart Selman and Carla P Gomes. 2006. Hill-climbing search. *Encyclopedia of cognitive science* 81 (2006), 82.

[73] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems* 31 (2018).

[74] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*.

PMLR, 6105–6114.
[75] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1021–1038.
[76] Nenad Tomašev, Xavier Glorot, Jack W Rae, Michal Zielinski, Harry Askham, Andre Saraiva, Anne Mottram, Clemens Meyer, Suman Ravuri, Ivan Protsyuk, et al. 2019. A clinically applicable approach to continuous prediction of future acute kidney injury. *Nature* 572, 7767 (2019), 116–119.
[77] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 26–49.
[78] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229* (2020).
[79] Jiachuan Wang, Lei Chen, and Charles Wang Wai Ng. 2022. A new class of polynomial activation functions of deep learning for precipitation forecasting. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 1025–1035.
[80] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. 2022. Piranha: A {GPU} Platform for Secure Computation. In *31st USENIX Security Symposium (USENIX Security 22)*. 827–844.
[81] Runhua Xu, Nathalie Baracaldo, and James Joshi. 2021. Privacy-preserving machine learning: Methods, challenges and directions. *arXiv preprint arXiv:2108.04417* (2021).
[82] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 162–167.
[83] Yuanhan Zhang, ZhenFei Yin, Yidong Li, Guojun Yin, Junjie Yan, Jing Shao, and Ziwei Liu. 2020. Celeba-spoof: Large-scale face anti-spoofing dataset with rich annotations. In *European Conference on Computer Vision*. Springer, 70–85.

A RELATED WORK ON COMPLEX ACTIVATION FUNCTIONS

In this section, we review some work on activation functions (AFs) that fall outside the scope of our work — but nevertheless be interesting to readers.

Complex AFs used in early NN. Early neural network (NN) models used binary threshold units [22, 34] and subsequently sigmoid and TanH as AFs. The AFs are complex as well. However, research in the last decades has exhibited ReLU AF outperforms such complex used by primary NN models convincingly. Hence, in our work, we focus on more recent complex AFs that have superior performance than ReLU.

Linear splines based AFs. Methods presented in [4, 7, 8, 62] use a set of linear learnable splines as AFs in 1-Lipschitz constraint neural network. These linear splines are MCP-friendly. But 1-Lipschitz constraint neural network, due to being prone to vanishing gradients problem and having less expressiveness [49], is not generally used for cloud-based inference services. Thus, they fall outside the scope of our paper.

Polynomial AFs. Recently, another trendy line of work attempts to redesign AF with a polynomial function [2, 17]. During training, these new AF are used instead of the traditional AF. However, in many real-world scenarios, we assume the DNN model has already been trained over traditionally used complex AF, and retraining further is considered expensive and time-consuming. Hence, these works are outside the scope of our work as well.

Batch normalization and AF approximation. Prior works [10, 52, 79] that use batch normalization to generate an accurate approximation of activation function mostly focus on ReLU. Amongst these, only the work from Chabanne et al. [10] falls within the scope of this study. However, their "least square fit" approach with a single polynomial makes it unsuitable to render an approximation that has negligible accuracy loss.

The other two works, do not fall within the scope, as they require further re-training of the model over the approximated ReLU which does not satisfy our design goal ❹.

MiniONN (m, k, n, a, b):
// Pick n equally-spaced points in $[a, b]$
$X \leftarrow \{a + i \cdot (b-a)/n \mid i \in \{0, 1, \ldots, n\}\}$
$S \leftarrow \{a, b\}$
while $|S| \leq m + 2$ **do**
  $x^* \leftarrow \text{argmin}_{\{x \in X\}}\ \text{fitness}(X, S \cup \{x\}, \text{k})$
  $S \leftarrow S \cup \{x^*\}$
$\widehat{F}_{\text{act}} \leftarrow \emptyset$
for $i \in \{2, \ldots, |S|\}$ **do**
  $X' \leftarrow \{x \in X \,|\, S[i-1] \leq x \leq S[i]\}$
  $Y' \leftarrow \{F_{\text{act}}(x) \,|\, x \in X'\}$
  $f_i \leftarrow \text{spline}(X', Y', \text{k})$
  $\widehat{F}_{\text{act}} \leftarrow \widehat{F}_{\text{act}} \bigcup \{f_i\}$
return $\widehat{F}_{\text{act}}$

fitness(X, S, k) : // Measures goodness of the fit
// Sort S in ascending order. Also ensure $S[1] = a$ and $S[|S|] = b$
for $i = 2, \ldots, |S|$ do
  $X' \leftarrow \{x \in X \,|\, S[i-1] \leq x \leq S[i]\}$
  $Y' \leftarrow \{F_{\text{act}}(x) \,|\, x \in X'\}$
  $f_i \leftarrow \text{polyfit}(X', Y', \text{k})$
  $\delta_i \leftarrow \sum_{x \in X'} \|F_{\text{act}}(x) - f_i(x)\|_2$
return $\sum_i \delta_i$

[§]Similar to Liu et al. [51], we use `scipy.interpolate.UnivariateSpline`, and `numpy.polyfit` libraries to implement Spline, and polyfit function respectively.

Figure 6: Recipe used by MiniONN [51] to approximate activation functions F_{act} between $[a, b]$ by a set of m piece-wise continuous splines $\widehat{F}_{\text{act}} = \bigcup_{i=1}^{\mathsf{m}} \{f_i\}$ such that $\forall i$ $\mathbf{Deg}(f_i) \leq \mathsf{k}$.

B DISCUSSION ON GRADIENT FREE LOCAL OPTIMIZATIONS

As discussed in Section 4.3, in this work, we try to solve the constraint optimization problem in Equation (10).

Our initial attempt was to solve Equation (10) using a hill climbing (HC) [72] type approach. However, it failed as HC tends to get stuck while solving optimization problems. We, therefore, choose a simulated annealing-based approach since it is scholastic and generally resistant to getting stuck during solution searches. One may explore other techniques from the existing gradient-free local optimization literature (e.g., ant colony optimization, tabu search, genetic algorithm, etc.) [20]. We believe they may work equally well given they are successfully customized for secure inference.

C DETAILS OF MINIONN

Liu et al. [51] proposed MiniONN, a scheme to to generate MPC-friendly piece-wise polynomials approximations for sigmoid, and tanh function. We implement this recipe ourselves to generate MPC-friendly approximations for complex activation functions (AFs) (e.g., SiLU, GeLU, Mish) between range $a = -5$, and $b = 5$ as shown in Figure 6

Briefly, this approach takes n equally spaced points between $[a, b]$ to approximate the given AF F_{act}. Let's denote the set of these points as $X \leftarrow \{x_1, x_2, \ldots, x_n\}$ where $x_1 = a$, and $x_n = b$, and

```
Input layer:
    784 x 50
    batch normalization1d (50)
    activation function()
Hidden layer 1:
    50 x 50
    batch normalization1d (50)
    activation function()
Layer 3: Output layer:
    50 x 10
    batch normalization1d (50)
    activation function()
```

Figure 7: Fully connected neural network with one hidden layer we use for DigitRecognition task.

$Y \leftarrow \{y_1, y_2, \ldots, y_n\}$ be the set of values used to approximate F_{act} where $y_i = F_{\text{act}}(x_i)$. Then MiniONN attempts to find a set of m switchover points $S \leftarrow \{s_1, s_2, \ldots s_m\}$ between $[a, b]$. The points from S are used as knots to approximate F_{act} using $m + 1$ MPC-friendly polynomials (same as Equation (4)); where f_i is a spline approximating the region between $\{s_i, s_{i+1}\}$ for $i \in \{1, 2, \ldots m-1\}$. Note $f_0 = 0$, and $f_{m+1}(x) = x$. MiniONN finds these m switching points by iterating over each $x \in X$ and selecting a new point for iteration that maximizes the overall goodness of the fit.

For our case, we consider squared mean error as way to measure the goodness of the fit — as shown in the fitness procedure (Figure 6). We explored a few parameters for $\langle n, m, k\rangle$ and settle for $n = 1,000$, $m = 20$, and k = 3 since it yields the best accuracy as reported in Table 3.

D DNN MODELS CONSIDERED FOR DIFFERENT TASKS

For experimental evaluation, in this work, we consider four DNN models for four image classification tasks. For DigitRecognition task, we consider a fully connected neural network (FCN) with one hidden layer as shown in Figure 7. For the second task CIFAR10Classification, we consider a convolutional neural network (CNN) with five hidden layers (Figure 8). Since the datasets we consider for the last two tasks are relatively more complex, we select complex DNN models for them.

In particular, for ImageNet1KClassification, we consider a residual neural network (ResNet9) having eight hidden layers (Figure 9). For SpoofFaceDetection task we choose the EfficientNetB0 model having 17 hidden layers (Figure 10). We refer the interested reader to [30, 74] for more details. Reader should observe that inputs to the activation function are batch normalized for each of the four models – a standard practice in DNN models.

E 2PC RESULTS

We use the oblivious transfer-based construction for CryptFlow2. We observe a similar performance gain as the 3PC scenario — 2×-5× speedup of Compact compared to NFGen— as shown in Table 5.

F MEMBERSHIP INFERENCE ATTACKS

Membership inference (MI) is a popular way to examine the privacy of the training data and predicts if a given input (x, y) was used to train a model f_θ. We focus on the state-of-the-art MI attack by Carlini et al. [9].

To find if a specific input value (x, y) was used to train a model, the attacker first creates N samples of training data from the data distribution $\mathbb{D}$, such that half of the datasets contain (x, y) in them,

```
Input layer:
    conv2d (3, 32); Kernel size = 3 ; Padding - 1
    batch normalization (32)
    activation function
Hidden layer 1:
    conv2d (32, 64); Kernel size = 3; Padding - 1; Stride = 1
    batch normalization (64)
    activation function
    max pool (2,2)
Hidden layer 2:
    conv2d (128, 256); Kernel size = 3; Padding - 1; Stride = 1
    batch normalization (128)
    activation function
Hidden layer 3:
    conv2d (256, 256); Kernel size = 3; Padding - 1; Stride = 1
    batch normalization (256)
    activation function
    max pool (2,2)
Hidden layer 4:
    Linear (256 x 4 x 4, 1024)
    Batch normalization (1024)
    activation function
Hidden layer 5:
    Linear (1024, 512)
    Batch normalization (512)
    activation function
Output layer:
    Linear (512, 10)
    Softmax (dim=1)
```

Figure 8: Convolutional neural network (CNN) model with 5 hidden layers we use for CIFAR10Classification task.

Task Name	# HLs†	F_{act}	NFGen	Ours	**Speedup**
DigitRecognition	1	Mish	184	170	1.08×
		SiLU	290	305	0.95×
		GeLU	135	133	1.01×
CIFAR10Classification	5	SiLU	723	205	3.52×
		GeLU	745	165	4.50×
		Mish	825	229	3.60×
ImageNet1KClassification	8	SiLU	512	122	4.20×
		GeLU	502	108	4.64×
		Mish	537	147	3.63×
SpoofFaceDetection	17	SiLU	827	189	4.37×
		GeLU	876	192	4.54×
		Mish	893	203	4.38×

† # HLs = Number of hidden layers. We experiment with the oblivious transfer (OT) based construction of CryptFlow2.

Table 5: Comparison of inference time (ms) of three activation functions (F_{act}) over four different classification tasks for $N = 2$ servers using CryptFlow2 MPC library. Since the DNN model used in DigitRecognition task has only one hidden layer (# HLs=1), the performance of NFGen task is similar to Compact. However, as DNN models become more complex and deep, having high hidden layers; for the other three tasks Compact outperforms NFGen — exhibiting a speedup 2×–5× when compared to NFGen.

Input layer::
- conv2d (3, 64); Kernel size = 3 ; Padding = 1
- batch normalization (32)
- activaton function

Hidden layer 1:
- conv2d (64, 128); Kernel size = 3; Padding = 1
- batch normalization (64)
- activaton function
- max pool (2,2)

Hidden layer 2:
- conv2d (128, 128); Kernel size = 3; Padding = 1
- batch normalization (128)
- activaton function

Hidden layer 3:
- conv2d (128, 128); Kernel size = 3; Padding = 1
- batch normalization (128)
- activaton function

Hidden layer 4:
- conv2d (128, 256); Kernel size = 3; Padding - 1; Stride = 1
- batch normalization (256)
- activaton function
- max pool (2,2)

Hidden layer 5:
- conv2d (256, 256) Kernel size = 3; Padding - 1; Stride = 1
- Batch normalization (512)
- activaton function

Hidden layer 6:
- conv2d (256, 512) Kernel size = 3; Padding - 1; Stride = 1
- Batch normalization (512)
- activaton function

Hidden layer 7:
- conv2d (512, 512) Kernel size = 3; Padding - 1; Stride = 1
- Batch normalization (512)
- activaton function

Hidden layer 8:
- conv2d (512, 512) Kernel size = 3; Padding - 1; Stride = 1
- Batch normalization (512)
- activaton function

Output layer:
- Max pool
- Flatten ()
- Dropout
- Linear (512, 100)
- Softmax (dim=1)

Figure 9: ResNet9 [30] DNN architecture with eight hidden layers we use for ImageNet1KClassification task.

Task Name	F_{act}	m	k	$\mathcal{R}$
DigitRecognition	SiLU	78	3	$\langle 64, 32\rangle$
	GeLU	82	5	$\langle 64, 32\rangle$
	Mish	85	3	$\langle 64, 32\rangle$
CIFAR10Classification	SiLU	77	3	$\langle 84, 42\rangle$
	GeLU	34	3	$\langle 84, 63\rangle$
	Mish	93	4	$\langle 84, 42\rangle$
ImageNet1KClassification	SiLU	91	5	$\langle 84, 42\rangle$
	GeLU	101	5	$\langle 84, 42\rangle$
	Mish	96	5	$\langle 84, 42\rangle$
SpoofFaceDetection	SiLU	81	5	$\langle 64, 32\rangle$
	GeLU	90	5	$\langle 64, 32\rangle$
	Mish	93	5	$\langle 64, 32\rangle$

Table 6: m, k, $\mathcal{R}$ for four tasks using FindBestPiecePoly. We consider accuracy loss $v < 10^{-2}$ as negligible. $\mathcal{R}$ is presented as $\langle \ell, d\rangle$.

and other half do not. Then the attacker trains two sets of shadow models M_1 (where the datasets contain (x, y)) and M_2 (where the

MBConv1 (K x K, B, S):
- Depthwise Conv (K x K, M, S)
- Batch normalization
- Activation function
- SE (R=4)
- Conv (1 x 1, B, 1)
- Batch normalization

MBConv6 (K x K, B, S):
- Conv (1 x 1, 6M, 1)
- Batch normalization
- Activation function
- Depthwise Conv (K x K, 6M, S)
- Batch normalization
- Activation function
- SE (R = 4)
- Conv (1 x 1, B, 1)
- Batch normalization

Input layer: conv2d (3, 3); Kernel size = 32 x 32 x 3
Hidden layer 1: MBConv1 (3, 3); Kernel size = 16 x 16 x 32
Hidden layer 2: MBConv6 (3, 3); Kernel size = 16 x 16 x 16
Hidden layer 3: MBConv6 (3, 3); Kernel size = 8 x 8 x 24
Hidden layer 4: MBConv6 (3, 3); Kernel size = 8 x 8 x 24
Hidden layer 5: MBConv6 (3, 3); Kernel size = 4 x 4 x 40
Hidden layer 6: MBConv6 (3, 3); Kernel size = 4 x 4 x 40
Hidden layer 7: MBConv6 (3, 3); Kernel size = 4 x 4 x 80
Hidden layer 8: MBConv6 (3, 3); Kernel size = 4 x 4 x 80
Hidden layer 9: MBConv6 (3, 3); Kernel size = 4 x 4 x 80
Hidden layer 10: MBConv6 (3, 3); Kernel size = 2 x 2 x 112
Hidden layer 11: MBConv6 (3, 3); Kernel size = 1 x 1 x 112
Hidden layer 12: MBConv6 (3, 3); Kernel size = 1 x 1 x 112
Hidden layer 13: MBConv6 (3, 3); Kernel size = 1 x 1 x 112
Hidden layer 14: MBConv6 (3, 3); Kernel size = 1 x 1 x 192
Hidden layer 15: MBConv6 (3, 3); Kernel size = 1 x 1 x 192
Hidden layer 16: MBConv6 (3, 3); Kernel size = 1 x 1 x 192
Hidden layer 17: MBConv6 (3, 3); Kernel size = 1 x 1 x 320
Output layer: Max pool, Flatten, Dropout, Linear (512, 100), Softmax (dim=1)

Figure 10: EfficientNetB0 [74] DNN architecture with 17 layers we use for SpoofFaceDetection task.

datasets do not contain (x, y)). $\mathbb{Q}_{\text{in}}$ denotes the distribution of losses of (x, y) from M_1 and $\mathbb{Q}_{\text{out}}$ is the distribution of the cross losses from M_2. Finally, the attacker calculates the cross entropy loss of (x, y) on the target model $\ell(f_\theta(x), y)$ and measure the likelihood of this loss under the distributions $\mathbb{Q}_{\text{in}}(x, y)$ and $\mathbb{Q}_{\text{out}}(x, y)$ and return whichever is more likely.

Thus, any approach [56, 59] (including ours) leveraging training data (or a holdout data) to find an approximate AF could affect $\ell(f_\theta(x), y)$, $\mathbb{Q}_{\text{in}}$, and $\mathbb{Q}_{\text{out}}$, and thus potentially has an impact on MI attack. However, since all these prior works have negligible accuracy loss over the testing data which is sampled from the distribution $\mathbb{D}$ (i.e., in-distribution-data), we conjecture that there is a chance that $\ell(f_\theta(x), y)$, and $\mathbb{Q}_{\text{in}}$ or $\mathbb{Q}_{\text{out}}$ remain mostly similar when the given input $\{x, y\}$ is sampled from the in-distribution data, and thus may not affect MI attack.

When the given input $\{x, y\}$ is sampled from outside the in-distribution data (i.e., outliers), it is difficult to assess how $\ell(f_\theta(x), y)$, $\mathbb{Q}_{\text{in}}(x, y)$ or $\mathbb{Q}_{\text{out}}(x, y)$ would get affected. Interestingly, Carlini et al. [9] briefly discuss that outliers are inherently more vulnerable to their MI attack. This is because, as they experimentally show, the gap between $\mathbb{Q}_{\text{in}}$ and $\mathbb{Q}_{\text{out}}$ widens when outliers are inserted into the training dataset of f_θ [9, Figure 11], and this enables the attacker to make predictions about $\{x, y\}$ with higher accuracy.

In summary, how $\ell(f_\theta(x), y)$, and $\mathbb{Q}_{\text{in}}$, or $\mathbb{Q}_{\text{out}}$ are affected for both in-distribution, and outlier data due to the introduction of secure inference or training protocols, and whether such secure protocols can be used to defend against MI attacks is an interesting open question.