SAFENET: A SECURE, ACCURATE AND FAST NEURAL NETWORK INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

The advances in neural networks have driven many companies to provide prediction services to users in a wide range of applications. However, current prediction systems raise at least one party's privacy concern: either the user's private data or the service provider's proprietary neural network model. A cryptographic neural network inference service is an efficient way to allow two parties to execute neural network inference without revealing either party's data or model. Nevertheless, existing cryptographic neural network inference services suffer from huge running latency. Especially, the latency of communication-expensive cryptographic activation function is 3 orders of magnitude higher than plaintext-domain activation function. Activations are the necessary components of the modern neural networks. Therefore, slow cryptographic activation has already become the obstacle of efficient cryptographic inference.

In this paper, we propose a new technique, called SAFENet, to enable a Secure, Accurate and Fast nEural Network inference service. To speedup secure inference and guarantee inference accuracy, SAFENet includes channel-wise activation approximation with multiple-degree options. This is implemented by keeping the most useful activation channels and replacing the remaining less useful channels by the various-degree polynomials. SAFENet also supports mixed-precision activation approximation by automatically assigning different replacement ratios to various layers. So mixed precision method further increases the approximation ratio and reduces inference latency. Our experimental results show SAFENet obtains the state-of-the-art inference latency without accuracy decrease, reducing $38\% \sim 61\%$ latency over the prior techniques on various encrypted datasets.

1 INTRODUCTION

Neural network inference as a service is an effective method for users to use intelligent services on powerful servers. Neural network models are efficiently trained on the big organization's proprietary data. There are many emerging intelligent client-server applications like smart speakers, voice assistants, and image classifications Mishra et al. (2020). However, multi-party based neural network inference services raise privacy concerns. To complete the intelligent service, either the users need to upload their input data to the network holders, or the model holders need to release neural networks to the users for the local inference. The former violates the privacy of users, while the latter exposes the service provider's proprietary model, which potentially make other competitors steal the proprietary model so that model holders cannot monetize the trained models.

Modern cryptography such as Homomorphic Encryption (HE) and Multi-Party Computation (MPC) enables secure inference services that protect both service provider's proprietary model and the user's private data. During secure inference services, the provider's confidential model is not released to any users and the user's private data is encrypted by HE or MPC. CryptoNets Gilad-Bachrach et al. (2016) is the first HE-based secure neuron network on encrypted data. However, it is far from practical since huge computational overhead. For example, CryptoNets takes ~ 298 seconds to perform one secure MNIST image inference on a powerful server. MiniONN Liu et al. (2017) and Gazelle Juvekar et al. (2018) prove that using the hybrid of HE and MPC is able to design low-latency secure inference. Although Gazelle significantly reduces the MNIST inference latency of CryptoNets into ~ 0.3 seconds by using the hybrid of HE and MPC, it is still far from practical on larger dataset such as CIFAR-10 and CIFAR-100, because of heavy HE encryption protocol and

expensive operations to support the non-linear activation function in a deep neural network. For instance, Gazelle requires ~ 240 seconds latency and ~ 8.3 GB communication to perform ResNet-32 on CIFAR-100 dataset Mishra et al. (2020). NASS Bian et al. (2020), CONAD Shafran et al. (2019) and CryptoNAS Ghodsi et al. (2020) are proposed to design cryptography-friendly neural network architectures, but they still suffer from heavy encryption protocol in the online phase. Delphi Mishra et al. (2020) significantly reduces the online latency by moving most heavy cryptography computations into offline phase. The offline computation can be pre-processed in advance.

However, the state-of-the-art cryptographic inference service Delphi still suffers from huge online latency. This is because a big communication overhead between the user and the service provider is required to support cryptographic $ReLU$ activations. Our experiments show that the communication overhead is proportional to $ReLU$ units in the whole neural network. Delphi tries to reduce inference latency by replacing expensive $ReLU$ with cheap polynomial approximation. Unfortunately, most $ReLU$ units are hard to be replaced without accuracy decrease by Delphi. The accuracy will be dramatically decreased if more $ReLU$ units are approximated by polynomials. Specifically, Delphi only replaces $\sim 42\%$ $ReLU$ numbers on a CNN-7 network (MiniONN Liu et al. (2017) detailed in Section 6.2) and $\sim 20\%$ $ReLU$ numbers on ResNet-32 network with $< 1\%$ accuracy decrease. When Delphi approximates more $ReLU$ units, $> 3\%$ inference accuracy will be lost compared to all-$ReLU$ model. If accuracy loss is constrained, non-linear layers still occupy almost 62% to 74% totol latency in CNN-7 and ResNet-32 networks. Therefore, slow non-linear layers are still the obstacle of fast and accurate secure inference.

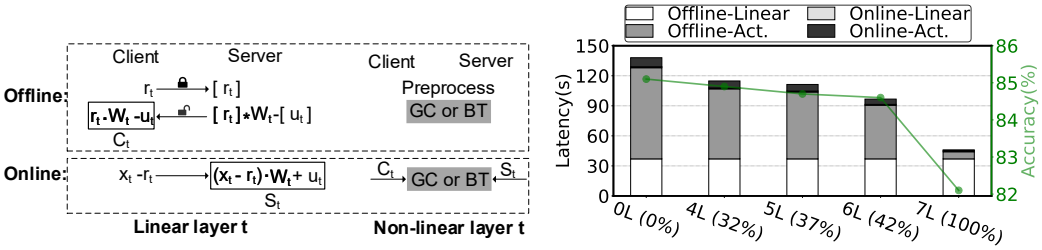
Our contribution. One key observation is that the layer-wise activation approximation strategy in Delphi is too coarse-grained to replace the *bottleneck layers* in which the $ReLU$ units are mainly located, e.g. the first layer in CNN-7 occupies $> 58\%$ $ReLU$ units. The channels in *bottleneck layers* are hard to be totally replaced without small accuracy loss. To meet accuracy constraints and speedup secure inference, SAFENet includes a more fine-grained channel-wise activation approximation to keep the most useful activation channels within each layer and replace the remaining less important activation channels by polynomials. In this way, only partial channels in each layer will be approximated, which is approximate-friendly for *bottleneck layers*. Another contribution of SAFENet is that automatic multiple-degree polynomials exploration in each layer is supported, compared to prior works using only degree-2 polynomials. Also, SAFENet enables mixed-precision activation approximation by assigning different approximation ratios to various layers, which further replaces more $ReLU$ units with cheap polynomials. Our results show that under the same accuracy constraints, SAFENet obtains the state-of-the-art inference latency, reducing $38\% \sim 61\%$ latency over the prior techniques.

2 Background and Related Work

Threat Model and Cryptographic Primitives. Our threat model is the same as previous work Delphi Mishra et al. (2020). More specifically, we consider the service holder as a semi-honest cloud which attempts to infer clients input information but follows the protocol. Server holds the Convolutional Neural Network (CNN) model and client holds the input to the network. For linear computations, the client encrypts input and sends it to the server using a HE scheme Mishra et al. (2020), and then server returns encrypted output to the client. The client decrypts and decodes the received output. The secret sharing (SS) in Delphi is used to protect the privacy of intermediate results in the hidden layers. Then garbled circuits (GC) guarantees the data privacy in the activation layers, like $ReLU$, and secret sharing is used to securely combine HE and GC. Other than GC, Beaver's multiplicative Triples (BT) proposed by Beaver (1995) can be used to implement approximated activation using secure polynomials. BT-based polynomial approximation for $ReLU$ is 3-order magnitude cheaper than GC-based $ReLU$ units in average, so it is used to design approximated secure activation function. At the end of secure inference, server learns nothing but client learns the inference result. More details of cryptographic primitives can be found at appendix A.1.

2.1 Cryptographic Inference.

Modern neural networks usually consist of linear convolution layers and non-linear activation layers. As Figure 1a shows, current state-of-the-art cryptographic inference, Delphi, has an offline phase and an online phase. The offline phase is independent of input data and is used to prepare data for the subsequent online phase. Each phase also has linear operations and non-linear operations.



(a) The cryptographic inference in our baseline Delphi. (b) Various approximate layers number (ratio).

Figure 1: Approximated ratio ablation study of CNN-7 on CIFAR-10 under 84.5% accuracy.

1. Offline linear layer. During offline linear process, the client samples a random matrix r_t that has the same shape with private input x_t, and then sends its encryption $[r_t]$ to server. The server processes homomorphic convolutions and returns $[r_t] * W_t - [u_t]$ to client, where W_t is the t-$layer$ network weights and $[u_t]$ is a ciphertext of sampled matrix by server. The last step in the offline linear layer is that client obtains $C_t = r_t \cdot W_t - u_t$ which is one part of secret sharing of $x_t \cdot W_t$.

2. Online linear layer. The online linear phase aims to let server obtain $S_t = (x_t - r_t) \cdot W_t + u_t$ which is the other secret sharing part of $x_t \cdot W_t$. It is almost as fast as the unencrypted computation, since the online input is a plaintext $x_t - r_t$.

3. Offline Layer-Wise Activation layer. Delphi supports layer-wise activation function where each activation layer either is $ReLU$ based on GC or is the approximated polynomial based on BT. During the offline phase, GC needs to generate and share the garbled circuits. BT needs to generate and share the Beaver's triples.

4. Online Layer-Wise Activation layer. During the online phase of layer-wise activation, $ReLU$ is either performed by GC or the approximated degree-2 polynomial. The latency of approximated activation implemented by BT is 192× smaller than $ReLU$ based on GC.

Latency Bottleneck and Motivation. Figure 1a shows our baseline Delphi suffers from long latency and low accuracy under coarse-grained layer-wise activation approximation. Specifically, $0L$ in Figure 1b means none of $ReLU$ layers and 0% $ReLU$ units are approximated by polynomials. The activation latency takes 72.5% of total latency. For only online phase, activation latency occupies $\sim 99\%$ of online latency. Therefore, activation layers are the performance bottleneck. $0L$ also shows all-$ReLU$ model achieves 85.1% accuracy. With increased approximation layer numbers, Delphi is able to improve the approximation ratios, but the inference accuracy is decreased at the same time. With the 84.5% accuracy constraints, Delphi at most replaces 6-layer $ReLU$ layers, with an approximation ratio of $\sim 42\%$ $ReLU$ units. Figure 2 shows the reason why approximation ratio is so low (42%). It is difficult to totally replace the $ReLU$ units by polynomials in the $bottleneck$ layer that has $> 58\%$ $ReLU$ units without large accuracy decrease. The layer-wise activation approximation is too coarse-grained. If Delphi aims to replace $> 42\%$ $ReLU$ units, the first $ReLU$ layer has to be replaced by polynomials, which brings significant accuracy decrease. The $ReLU$ units in modern networks are mainly located in the first few layers, and the $ReLU$ numbers are usually decreased exponentially shown in Figure 2. Especially for the much deeper neural networks on large dataset, replacing the first $ReLU$ layer significantly decreases the accuracy. To solve above problems, we propose a more fine-grained channel-wise activation approximation method which is modelled as a hyper-parameter optimization problem.

2.2 Population Based Training (PBT).

Inspired from evolutionary algorithm, Population Based Training (PBT) proposed by Jaderberg et al. (2017) is a more efficient method to jointly optimize model weights and user-specified hyper-parameters automatically during training. Many Reinforcement Learning (RL) based hyper-parameter optimization algorithms (Wang et al. (2019) and Lou et al. (2020)) are not able to efficiently optimize hyper-parameters, since they simply stop training prematurely and consider partially trained accuracy as the final accuracy or reward. PBT is computationally efficient since it adapts parallel scheme and weight sharing in the evolutionary process. Specifically, given a pre-trained model, many workers are created and each worker shares same pre-trained weights and has unique hyper-parameters. Then multiple workers are independently trained for several iterations and are evaluated by the user-specified score function. In the exploitation, workers with better scores will keep their parameters, and copy their weights and hyper-parameters to the other workers. Workers

Features	Gazelle	NASS	CONAD	Delphi	CryptoNAS	SAFENet
Optimized Activation	✗	✗	✓	✓	✓	✓
Channel-Wise	✗	✗	✗	✗	✗	✓
Mixed-Precision	✗	✗	✗	✗	✗	✓
Multiple-Degrees	✗	✗	✗	✗	✗	✓

Table 1: Cryptographic inference works.

Figure 2: Neurons ratio.

with worse scores will perform exploration to randomly scale their hyper-parameters. The training, scoring, exploitation and exploration repeat until the score function is convergent. In this paper, we model the channel-wise activation approximation task as a hyper-parameter optimization problem.

2.3 Comparison with prior works.

Table 1 shows a comparison between prior works and SAFENet. Gazelle Juvekar et al. (2018) and MiniONN prove the feasibility of the hybrid use of GC and HE, but they both suffer from huge latency. NASS Bian et al. (2020), CONAD Shafran et al. (2019), and CryptoNAS Ghodsi et al. (2020) try to improve Gazelle and MiniONN's performance by the co-design of neural network architectures and cryptographic protocol, but they all require a heavy online cryptographic phase. Delphi Mishra et al. (2020) reduces online latency by moving partial online operations into offline phase and replacing layer-wise activation by degree-2 polynomials. Only our SAFENet supports more fine-grained channel-wise activation approximation with multiple-degree polynomials exploration, shown in Table 1. SAFENet also enables the mixed-precision approximation ratios for different layers. Other works like XONN Riazi et al. (2019) and soteria Aggarwal et al. (2020) focus on GC-based binary neural networks, thereby suffering from inference accuracy loss.

3 SAFENet

Overview. In order to replace expensive non-linear activations by cheap linear polynomial approximations as much as possible, we mainly use three methods. Firstly, we use fine-grained activation approximation in a channel-wise manner, instead of coarse-grained approximation in a layer-wise manner. Secondly, since different layers in neural networks have different contributions on the inference accuracy, we propose an automatic mixed-precision method to assign a proper approximation ratio for each layer. According to the assigned approximation ratio, SAFENet replaces the corresponding channels in the ascending order of their channel important factors. Mixed-precision further improves the total approximation ratio, given an accuracy threshold. Moreover, SAFENet's mixed-precision planner explores different polynomial approximations with various degrees.

SAFENet Methods Definition. Given an T-Layer CNN model, we define that $\{W_t\}_{t=0}^{T-1}$ and $\{A_t\}_{t=0}^{T-1}$ are the model weights and activations, and let $W_t \in \mathbb{R}^{c_{out}\times c_{in}\times w_w\times h_w}$ and $A_t \in \mathbb{R}^{c_{in}\times w_a\times h_a}$ be the t-th layer model weights and activations, where c_{out} is output channel number, c_{in} is input channel number, w_w and h_w are kernels' width and height, w_a and h_a are activations' width and height. The t-th layer output feature maps $O_t \in \mathbb{R}^{c_{out}\times w_o\times h_o}$ can be derived by Equation 1, where $O_{t,i,:,:}$ is the output feature map of t-th layer i-th channel, $A_{t,j,:,:}$ is the t-th layer and j-th channel activation, $W_{t,i,j,:,:}$ is the model parameters of t-th layer, i-th output channel and j-th input channel, w_o and h_o are output feature's width and weight, and $*$ is the convolution operation.

$$O_{t,i,:,:} = \sum_{j=1}^{c_{in}} A_{t,j,:,:} * W_{t,i,j,:,:}. \tag{1}$$

Then the channel-wise activation approximation can be defined as Equation 2, where (t+1)-th layer activation A_{t+1} of i-th channel is derived by expensive $ReLU$ when the corresponding mask matrix $m_{t+1,i} = 0$, otherwise the approximated activation.

$$A_{t+1,i,:,:} = \begin{cases} ReLU(O_{t,i,:,:}), & \text{if } m_{t+1,i} = 0. \\ Approx(O_{t,i,:,:}), & \text{if } m_{t+1,i} = 1. \end{cases} \tag{2}$$

$$m_{t+1,i} = \begin{cases} 0, & \text{if } imp_order(A_{t+1,i}) \geq n_{t+1} \times (1-\alpha_{t+1}). \\ 1, & \text{Otherwise.} \end{cases} \tag{3}$$

The Equation 3 defines the mask matrix $m_{t+1,i}$. Here, $imp_order(A_{t+1,i}) \in [1, c_{out}]$ is the importance ascending order of i-th channel $A_{t+1,i}$ in the layer (t+1), and $imp_order(A_{t+1,i})$ can be derived by $sort(\sum_{j=1}^{C_{in}} |W_{t,i,j,:,:}|)$, where $\sum_{j=1}^{C_{in}} |W_{t,i,j,:,:}|$ is the L-1 norm of i-th channel weight, and this L-1 norm is proven effective as the channel importance of $A_{t+1,i}$ by prior works Li et al. (2017) and Liu et al. (2019). n_{t+1} in Equation 3 is the neuron number in (t+1)-th layer and α_{t+1} is the activation approximation ratio in the (t+1)-th layer. Equation 3 shows that channel-wise activation approximation in the (t+1)-th layer only approximate α_{t+1}-ratio less important activations with polynomials by setting their corresponding mask matrix entry $m_{t+1,i} = 1$. $Approx()$ activation function is enabled by multiple options including degree-3 polynomial $a_1x^3 + a_2x^2 + a_3x + a_4$, degree-2 polynomial $b_1x^2 + b_2x + b_3$, and degree-0 pruning, where a_1 to a_4 and b_1 to b_3 values are derived from the model training. The reason why we choose these three options is that our experiments show larger degree (> 3) polynomial approximation is hardly convergent in training because of the uncontrollable gradient exploding, and smaller degree (< 2) polynomial approximation shows worse performance. However, the proper insertions of degree-0 pruning may help on gradient exploding in some layers. We use hyper-parameter $\beta_t \in \{0, 2, 3\}$ as the polynomial degree in the $Approx()$ function. **Channel-Wise Activation Approximation** means T activation approximation ratios $\{\alpha_t\}_{t=0}^{T-1}$ are designed in a hand-crafted manner with $\beta_t = 2$. **Mixed-Precision Activation Approximation** automatically search the optimal combination of $\{\alpha_t\}_{t=0}^{T-1}$ using our improved PBT algorithm called BTPBT. BTPBT is described in Algorithm 1. **Multiple-degree approximation** means that approximation options $\{\beta_t\}_{t=0}^{T-1}$ are searched with α_t by BTPBT algorithm.

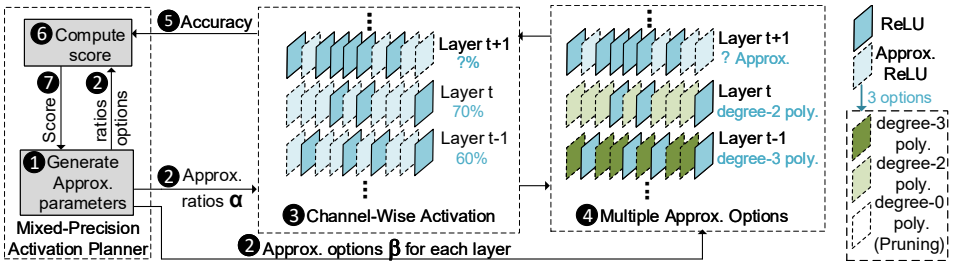


Figure 3: The overview of SAFENet working scheme. SAFENet uses a planner to automatically decide the channels approximation ratio of each layer and channel replacing methods including original $ReLU$, polynomial approximation and even pruning.

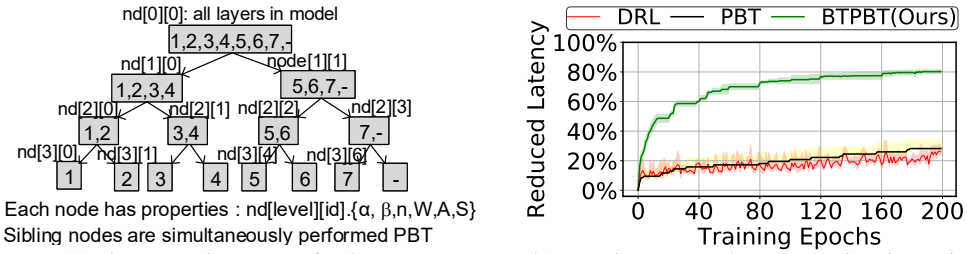
Working Flow. Figure 3 depicts the working flow of SAFENet. ❶ At first, the mixed-precision activation planner randomly initiates approximation parameters, $\{\alpha_t\}_{t=1}^{T-1}$ and $\{\beta_t\}_{t=1}^{T-1}$, to control channel-wise activation approximation and approximation options for each layer. ❷ Approximation parameters including $\{\alpha_t\}_{t=1}^{T-1}$ and $\{\beta_t\}_{t=1}^{T-1}$ are received by neural network model to decide which activation channels are approximated firstly, and then to decide which approximation method is picked up. ❸ In each t-th layer, $n_t \cdot \alpha_t$ unimportant channels are approximated by polynomials and $n_t \cdot (1 - \alpha_t)$ more important channels are kept as original activation function, e.g. $ReLU$, according to Equation 2 and Equation 3. ❹ The approximation methods for $n_t \cdot \alpha_t$ unimportant $ReLU$ units are decided. The approximation option consists of multiple-degree polynomials, including degree-β_t polynomials, where $\beta_t \in \{0,\ 2,\ 3\}$. ❺ The neural network model with channel-wise activation approximation is preformed for several times and outputs its inference accuracy. ❻ The mixed-precision activation planner uses the BTPBT method to compute the optimizing score defined in Equation 4, aware of accuracy and reduced latency at the same time. ❼ The generated score is used to guide the approximation parameters in the next iteration. These steps repeat until cryptographic inference accuracy and latency threshold are satisfied.

Score. The goal of our SAFENet is to reduce cryptography latency by replacing expensive original activation with cheap approximated activation as much as possible, given the user-defined accuracy threshold. Equation 4 is used to assign a $score$ for the channel-wise activation approximation parameters α_t and β_t. $\sum_{t=1}^{T} \alpha_t \times n_t$ is the number of $ReLU$ units that are replaced. $R(\beta_t)$ means the reduced time ratio using a degree-β_t polynomial to replace $ReLU$ units. For example, $R(\beta_t = 3) = 1 - \frac{0.2}{20.1} = 99.9\%$, since amortized degree-3 polynomial and $ReLU$ cost 0.2 us and 20.1 us, respectively. Therefore, $\sum_{t=1}^{T} \alpha_t \times n_t \times R(\beta_t)$ represents the reduced ratio of total latency. Our $score$ keeps the balance of accuracy and efficiency, enabling users to replace more activation neurons with multiple approximated polynomial degrees, while meeting the accuracy constraints.

$$score = accuracy \times (1 + \sum_{t=1}^{T} \alpha_t \times n_t \times R(\beta_t)) \tag{4}$$

Mixed-Precision Activation Planner. Finding appropriate parameters α_t and β_t to achieve higher score in equation 4 is a typical hyper-parameter optimization problem. Hand-crafted optimization for parameters is only practical to handle the search space of a single pair of α and β for all layers, which confines the improvement effect on activation approximation. Therefore, mixed-precision activation approximation is proposed to enable each layer to have unique α_i and β_i. However, applying hyper-parameter optimization methods to automatically search mixed-precision activation approximation is non-trivial. This is because of the huge search space of α_i and β_i for each layer. Assume $\alpha_i \in [0, 1]$ and it is discretized into the times of 0.05, therefore 21 options; $\beta_i \in \{0, 2, 3\}$, has 3 options. If a neural network model has T layers, the search space is $(3 \times 21)^T$. For this large search space, even PBT and Deep Reinforcement Learning (DRL) based methods (Wang et al. (2020)) suffer from slow learning speed and low approximation ratio. Figure 4b shows PBT and DRL methods cannot effectively reduce latency by mixed-precision activation approximation.

We propose Binary-Tree PBT algorithm (BTPBT) in Algorithm 1 to improve learning speed and further reduce secure latency. The high-level idea is to divide and conquer the complex mixed-precision search into a more easier semi-mixed-precision search. This is implemented by iteratively performing PBT in a binary tree order. Line $1 \sim 3$ show the process to construct a $(levels + 1)$-layer binary tree. Each node has properties $\{\alpha, \beta, n, W, A, S\}$ about approximation ratio, degree, total neuron numbers, model weights, accuracy and score. During nodes initialization, we define $\alpha = 0, \beta = 2$, and n, W, A parameters are derived from the pre-trained model, and S is acquired by equation 4. Figure 4a shows a binary tree example for the CNN-7 network. The root node, nd[0][0], consists of all neural networks layers, which means that its properties values, like α, are shared by all layers. In line 3 of algorithm 1, PBT function is used to update parameters of the root node. The lines from $4 \sim 8$ are used to perform PBT on each two siblings nodes simultaneously. If the current level has multiple siblings nodes, PBT is performed on these siblings nodes sequentially to avoid exploding search space. Line $6 \sim 8$ mean that the approximation neurons number of two siblings nodes are larger than their parent node. The lines from $9 \sim 10$ aim to filter the levels with lower accuracy than defined threshold $A_{threshold}$. Line 11 is used to select the the best candidate with the maximum score S. If we still need to improve the current candidates, line 12 takes the current best candidate as a good warm-staring point to global search better results.



(a) Binary tree in BTPBT for CNN-7.

(b) Learning curves about approximation ratio.

Figure 4: A binary tree example and the learning curve of CNN-7 on CIFAR-10.

Algorithm 1 BTPBT($A_{threshold}$)

1: $levels = log_2 T$
2: construct $(levels+1)$-layer binary tree and initialize all nodes nd[$level$][id].$\{\alpha, \beta, n, W, A, S\}$;
3: nd[0][0].$\{\alpha, \beta, n, W, A, S\} \leftarrow$ PBT(nd[0][0]);
4: **for** $i = 0,\ i < levels - 1,\ i++$ **do**
5: **for** $j = 0,\ j < 2^i,\ j++$ **do**
6: Limit child nodes' search space by nd[i][j].$\alpha\times$ nd[i][j].$n >$
7: nd[i+1][2j].$\alpha\times$ nd[i+1][2j].n + nd[i+1][2j+1].$\alpha\times$ nd[i+1][2j+1].n;
8: {nd[i+1][2j], nd[i+1][2j+1]}.$\{\alpha, \beta, n, W, A, S\}\leftarrow$PBT({nd[i+1][2j], nd[i+1][2j+1]});
9: **if** nd[i][$2^i - 1$].$A > A_{threshold}$ **then**
10: Store nd[i][$0 : 2^i - 1$] into candidate lists;
11: Select the entry with the maximum score S from candidates, denoted as nd[$level$][$0 : 2^{level-1}$];
12: **Return** nd[$level$][$0 : 2^{level-1}$].$\{\alpha, \beta, n, W, A, S\} \leftarrow$ PBT(nd[$level$][$0 : 2^{level-1}$]);

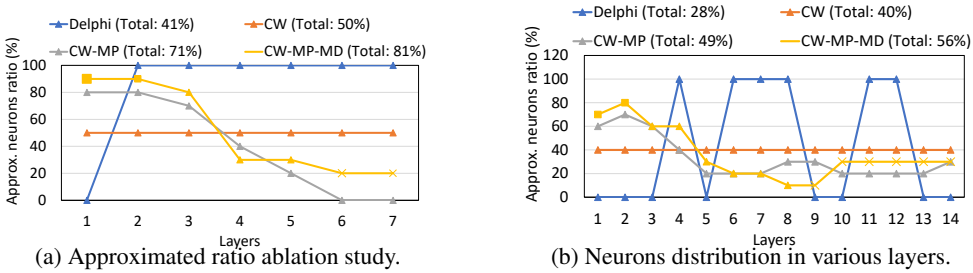
Figure 4 depicts the learning curves of PTB, DRL Wang et al. (2020) and our SAFENet which are used to search hyper-parameters α_i and β_i for the same score shown in equation 4. Different from PBT and DRL that directly search hyper-parameters for each layer, BTPBT firstly constructs a binary-tree structure shown in Figure 4a to iterativly search shared parameters by several layers. After 200 training epochs, BTPBT is able to replace $\sim 80\%$ neurons, but PBT and DRL only replace $< 30\%$ neurons. In the beginning of training, baselines and BTPBT are given a same pre-trained model. The DRL setting is specified in Wang et al. (2020). For PBT and our work, the evolution cycle between two exploitation is 20 iterations, and 50 workers are used to simultaneously search parameters. For BTPBT, each two sibling nodes and the root node require a PBT separately, therefore 2^{levels} PBT is required. We set 20 epochs for each above PBT. Also, we use 40 epochs to globally search the last-level nodes. The total 200 epochs with 50 workers take ~ 7 GPU hours.

4 Experimental Methodology

Benchmarks and Dataset. Our secure inference experiments rely on CIFAR-10 and CIFAR-100 datasets. Both CIFAR-10 and CIFAR-100 contain 50000 training images and 10000 testing images, where each image size is 32×32×3. Images in CIFAR-10 are classified into 10 typtes, but CIFAR-100 has 100 classes. Our baseline Delphi shows that CIFAR-100 is the most difficult dataset in the existing secure inference. We adopt CNN-7 specified in MiniONN Liu et al. (2017) to evaluate CIFAR-10. This CNN-7 architecture is also used in Delphi Mishra et al. (2020). Moreover, a deeper network VGG-16 Geifman (2017) is evaluated to further improve inference accuracy on CIFAR-10. For CIFAR-100, ResNet-32 He et al. (2016) is evaluated.

Systems Setup. We ran the secure inference models on two instances. They are equipped with an Intel Xeon E7-4850 CPU and 16 GB DRAM. The communication links between these two instances are in the LAN setting, and each instance uses 4 threads, same as the previous works. The hyper-parameters optimization for activation replacement requires an NVIDIA Tesla V100 GPU. The SEAL library SEAL and Multi-Protocol fancy-garbling library Carmer et al. (2019) are used to implement HE and garbled circuits functions. To avoid overflow, our experiments adopt 15-bit fixed point representation and the Least Significant Bits of intermediate results are also truncated to 15 bits. The hyper-parameters selection is implemented in python, and the BTPBT is constructed based on regular PBT Jaderberg et al. (2017) in Tune platform Liaw et al. (2018).

5 Results



(a) Approximated ratio ablation study. (b) Neurons distribution in various layers.

Figure 5: Approximated ratio ablation study of CNN-7 on CIFAR-10 under 84.5% accuracy.

Ablation Effects. Figure 5 describes the ablation effects of proposed techniques on CIFAR-10 with CNN-7 and VGG-16. CW, MP, and MD mean Channel-Wise, Mixed-Precisicion and Multiple Degrees, respectively. Square, triangle and cross marks in Figure 5 represent degree-3, degree-2 polynomial approximation and pruning, respectively. For the CNN-7 in Figure 5a, our baseline Delphi approximates $\sim 41\%$ activation by replacing the $ReLU$ in the last 6 layers with degree-2 polynomials. Channel-wise (CW) activation approximation using $\alpha = 0.5$ and $\beta = 2$ on all layers is able to achieve the same inference accuracy, while replacing more expensive activation neurons. Channel-wise under mixed precision (CW-MP) further improves the approximation ratio of the regular channel-wise method. Other than channel-wise approximation and mixed precision, Figure 5a shows that approximation ratio also benefits from exploring multiple degrees in various layers (CW-MP-MD). Figure 5b depicts the ablation study of a deeper network VGG-16. Our SAFENet with CW-MP-MD improves the approximation $ReLU$ numbers of Delphi by 28% under the same accuracy.

CNN-7	Offline		Online		Total	Accuracy
	HE-Linear	MPC-Act	HE-Linear	MPC-Act		
Gazelle	0s	91s	37s	9s	137s	85.1%
Delphi	37s	53.4s	0s	5.3s	95.4s	84.6%
SAFENet	37s	18.2s	0s	1.8s	53.4s	85.1%

Table 2: The CNN-7 result on CIFAR-10.

VGG-16	Offline		Online		Total	Accuracy
	HE-Linear	MPC-Act	HE-Linear	MPC-Act		
Gazelle	0s	142s	45s	16s	203s	89.6%
Delphi	45s	114s	0s	13s	172s	88.1%
SAFENet	40.2s	56.8s	0s	6.5s	104s	88.9%

Table 3: The VGG-16 result on CIFAR-10.

CIFAR-10. Table 2 shows the comparisons of CNN-7 implemented by prior techniques and SAFENet on CIFAR-10. Compared to Gazelle, Delphi not only moves the HE-based linear operations to the offline, but also reduces $\sim 41\%$ online and offline activation latency without significant accuracy loss. Our work SAFENet has similar latency with Delphi in the linear layer, but SAFENet reduces remarkable latency at non-linear activation layers. More specifically, SAFENet eliminates $\sim 66\%$ online and offline latency in activation layers, and reduces $\sim 61\%$ total latency with higher accuracy than Delphi. We also evaluated a deeper neural network VGG-16 on CIFAR-10 shown in Table 3. When Gazelle and Delphi have similar accuracy, Delphi only reduces $< 20\%$ latency of Gazelle on non-linear activation layer. This shows that Delphi benefits less from deeper neural networks. Compared to Gazelle, our work SAFENet is able to reduce $\sim 59.4\%$ online latency, 48.7% total latency. SAFENet has 50% less online accuracy and 39.5% less total latency with higher accuracy than Delphi, since SAFENet uses more find-grained channel-wise activation approximation.

ResNet-32	Offline		Online		Total	Accuracy
	HE-Linear	MPC-Act	HE-Linear	MPC-Act		
Gazelle	0s	158s	64s	18s	240s	67.9%
Delphi	64s	126.4s	0s	14.4s	204.8s	67.3%
SAFENet	57.6s	63.2s	0s	7.2s	128s	67.5%

Table 4: CIFAR-100 results using ResNet-32.

CIFAR-100. Table 4 shows the results of ResNet-32 evaluated on CIFAR-100. Gazelle achieves 67.9% inference accuracy, and each image inference consists of a 158-second offline phase and a 82-second online phase. Delphi reduces Gazelle's online latency by 67.6 seconds with only 0.4% accuracy decrease, but increases the offline latency by ~ 30 seconds. The offline phase is enlarged because Delphi moves most online HE operations to offline phase and only reduces few GC operations in offline. Compared to Gazelle, our work SAFENet is able to reduce both online and offline latency since we significantly reduce the number of GC-based activations. specifically, compared to Gazelle, SAFENet reduces 91.2% online latency and 23.5% offline latency. Compared to Delphi, SAFENet has 50% less online latency and 36.6% less offline latency. SAFENet reduces 46.7% and 37.5% of total latency over Gazelle and Delphi, respectively.

6 Conclusion

In this paper, we propose SAFENet to enable a Fast, Accurate and Secure neural Network inference service. SAFENet consists of three techniques including channel-wise activation, multiple-degree polynomials approximation, and mixed-precision approximation ratios. The channel-wise activation approximation keeps the most useful activation channels and replaces the remaining less useful channels with the various-degree polynomials. The mixed-precision activation approximation is implemented by assigning various layers with different approximation ratios. So the mixed precision method further increases the approximation ratio and reduces inference latency. Our experimental results show SAFENet obtains the state-of-the-art inference latency without accuracy decrease, reducing $38\% \sim 61\%$ latency over the prior techniques.

REFERENCES

Anshul Aggarwal, Trevor E Carlson, Reza Shokri, and Shruti Tople. Soteria: In search of efficient neural networks for private inference. *arXiv preprint arXiv:2007.12934*, 2020.

Donald Beaver. Precomputing oblivious transfer. In *Annual International Cryptology Conference*, pp. 97–109. Springer, 1995.

Song Bian, Weiwen Jiang, Qing Lu, Yiyu Shi, and Takashi Sato. Nass: Optimizing secure inference via neural architecture search. *arXiv preprint arXiv:2001.11854*, 2020.

Brent Carmer, Alex J. Malozemoff, and Marc Rosen. swanky: A suite of rust libraries for secure multi-party computation. `https://github.com/GaloisInc/swanky`, 2019.

Yonatan Geifman. Cifar vgg. `https://github.com/geifmany/cifar-vgg`, 2017.

Zahra Ghodsi, Akshaj Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. *arXiv preprint arXiv:2006.08733*, 2020.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pp. 201–210, 2016.

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

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL `http://arxiv.org/abs/1711.09846`.

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

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2017.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–631, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

Qian Lou, Feng Guo, Minje Kim, Lantao Liu, and Lei Jiang. Autoq: Automated kernel-wise neural network quantization. In *International Conference on Learning Representations*, 2020.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *USENIX Security*, 2020.

M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. {XONN}: Xnor-based oblivious deep neural network inference. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 1501–1518, 2019.

SEAL. Microsoft SEAL (release 3.2). `https://github.com/Microsoft/SEAL`, February 2019. Microsoft Research, Redmond, WA.

Avital Shafran, Gil Segev, Shmuel Peleg, and Yedid Hoshen. Crypto-oriented neural architecture design. *arXiv preprint arXiv:1911.12322*, 2019.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8612–8620, 2019.

Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

A APPENDIX

A.1 CRYPTOGRAPHIC PRIMITIVES.

Homomorphic Encryption (HE). Homomorphic Encryption (HE) is a cryptosystem that supports computation on ciphertexts x without decryption or private key. Given public key pk, private key sk, encryption function $\epsilon()$, and decryption function $\sigma()$, we can define an homomorphic operation $\times$ if there is an operation $\otimes$ such that $\sigma(\epsilon(x_1, pk) \otimes \epsilon(x_2, pk), sk) = \sigma(\epsilon(x_1 \times x_2, pk), sk)$, where x_1 and x_2 are sensitive plaintext from data holders, and only data holders have the private key sk.

Secret Sharing (SS) and Garbled Circuit (GC). Secret Sharing (SS) is usually based on additive sharing private values between the parties. For example, one variable x can be shared into two parts: $\langle x \rangle_1^A$ in P_1 and $\langle x \rangle_2^A$ in P_2, and variable x can be reconstructed by $x = \langle x \rangle_1^A + \langle x \rangle_2^A$. In previous secure inference schemes Juvekar et al. (2018), SS is used to protect the privacy of intermediate results in the hidden layers. Garbled Circuit (GC) is a cryptographic protocol that enables two parties (Garbler and Evaluator) to jointly compute a function over their private data without learning the other party's data. To use GC, the computed function should be represented into a Boolean circuit. The Garbler firstly garbles the Boolean circuit and generates the garbled table. The Evaluator receives the garbled table by the Oblivious Transfer Juvekar et al. (2018) and evaluates the table to the result. Previous secure inference schemes Juvekar et al. (2018) show that GC is used to process non-linear activation like $ReLU$ function.

Beaver's multiplicative Triples (BT). Beaver's multiplicative Triples proposed by Beaver (1995) can be considered as a secure two-party computation protocol. Assume two parties P_1 and P_2 have variables x and y, respectively. BT protocol enables two parties to obtain the secret sharing of product xy without revealing x and y. More details can be seen Mishra et al. (2020). BT-based polynomial approximation for $ReLU$ is 3-order magnitude cheaper than GC-based $ReLU$ units in average, so it is used to design approximated secure activation function.