AutoFHE: Automated Adaption of CNNs for Efficient Evaluation over FHE

Wei Ao   Vishnu Naresh Boddeti
Computer Science and Engineering
Michigan State University
{aowei, vishnu}@msu.edu

Abstract

Secure inference of deep convolutional neural networks (CNNs) under RNS-CKKS involves polynomial approximation of unsupported non-linear activation functions. However, existing approaches have three main limitations: 1) *Inflexibility:* The polynomial approximation and associated homomorphic evaluation architecture are customized manually for each CNN architecture and do not generalize to other networks. 2) *Suboptimal Approximation:* Each activation function is approximated instead of the function represented by the CNN. 3) *Restricted Design:* Either high-degree or low-degree polynomial approximations are used. The former retains high accuracy but slows down inference due to bootstrapping operations, while the latter accelerates ciphertext inference but compromises accuracy. To address these limitations, we present AutoFHE, which automatically adapts standard CNNs for secure inference under RNS-CKKS. The key idea is to adopt layerwise mixed-degree polynomial activation functions, which are optimized jointly with the homomorphic evaluation architecture in terms of the placement of bootstrapping operations. The problem is modeled within a multi-objective optimization framework to maximize accuracy and minimize the number of bootstrapping operations. AutoFHE can be applied flexibly on any CNN architecture, and it provides diverse solutions that span the trade-off between accuracy and latency. Experimental evaluation over RNS-CKKS encrypted CIFAR datasets shows that AutoFHE accelerates secure inference by 1.32× to 1.8× compared to methods employing high-degree polynomials. It also improves accuracy by up to 2.56% compared to methods using low-degree polynomials. Lastly, AutoFHE accelerates inference and improves accuracy by 103× and 3.46%, respectively, compared to CNNs under TFHE.

1 Introduction

MLaaS, machine learning as a service, is a rapidly growing market with many commercial offerings like Amazon Web Services (AWS), Google Google Cloud Platform (GCP), and Microsoft Azure. Its growth has been driven by the widespread success of deep learning on many tasks like vision [15, 24], language [14, 56], games [49, 51], science [17, 28], and many more. Figure 1 shows a typical MLaaS scenario. The Cloud (Alice) holds deep learning models, while the customer (Bob) has private data and requests service from Alice. Bob wants to protect his private data and does not want Alice to learn sensitive information. On the other hand, deep learning models, including *neural architectures* and *trained weights*, are properties of Alice. Alice spends considerable efforts to design neural architectures, like ResNets [24], ViT [15], and MLP-Mixer [55] and consumes huge computational resources to search for novel neural architectures [39, 60] or train network weights [22, 57].

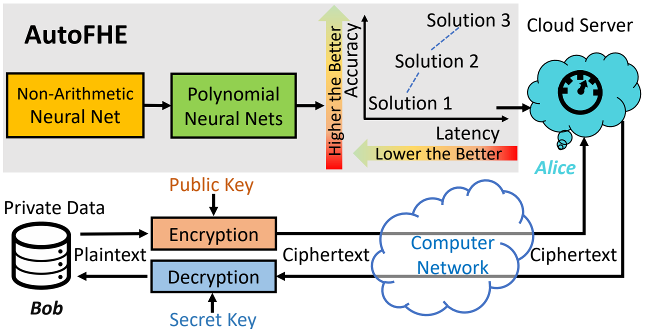

Figure 1: AutoFHE can automatically adapt the standard CNN with non-linear activations into a set of polynomial CNNs that span the trade-off between accuracy and latency for ciphertext inference. AutoFHE solutions can be deployed on the Cloud server to satisfy a range of customer requirements.

Homomorphic Encryption (HE): Secure inference of deep learning models under leveled homomorphic encryption (LHE) [6, 20, 41] or fully homomorphic encryption (FHE) [18, 33, 34] is a promising approach for resolving security concerns between Bob and Alice in the context of MLaaS. FHE enables us to evaluate a circuit with arbitrary depth, including modern deep CNNs [33]. Figure 1 shows secure inference of CNNs under FHE. First, Bob generates a public key to encrypt his private data and sends Alice the ciphertext. Second, Alice applies neural networks to process the ciphertext input, yielding an encrypted result. Finally, Bob uses the secret key to decrypt the encrypted result. Under FHE, Bob cannot learn Alice's neural architectures and weights, while Alice is also not exposed to Bob's data or the outcome.

Polynomial CNNs: Non-arithmetic activation functions, such as $\mathrm{ReLU}(x) = \max(x, 0)$, are a core component of modern CNNs, aiding in learning non-linear decision boundaries between classes. For example, residual networks (ResNets) [24] are composed of Conv-BN-ReLU triplets [24]. Since FHE only supports *multiplications* and *additions*, ReLU must be replaced by polynomial approximations to evaluate CNNs under FHE. Existing methods to generate polynomial CNNs fall into two categories; *manual design of low-degree and high-degree polynomial approximations*.

(1) A number of approaches adopt low-degree (typically ≤ 3) polynomials [6,12,20,41,42,45,46] to substitute non-arithmetic activation functions and *train* the resultant polynomial neural networks *from scratch*. For instance, CryptoNets [20], LoLa [6] and Delphi [45] employ a simple quadratic activation function x^2. Faster CryptoNets [12] exploit more accurate low-degree approximation $2^{-3}x^2 + 2^{-1}x + 2^{-2}$. SAFENet [42] adopts $a_1x^3 + a_2x^2 + a_3x + a_4$ or $b_1x^2 + b_2x + b_3$ and HEMET [41] uses $ax^2 + bx + c$. After low-degree polynomials are plugged into networks like ResNets both *network weights* and *polynomial coefficients* are trained from scratch using stochastic gradient descent (SGD). However, polynomial layers often lead to unstable training since they may dramatically amplify activations during forward propagation and gradients during backward propagation. For example, gradient explosion was observed in prior works [42,45]. As such, low-degree approaches suffer from a *dilemma*. On the one hand, since low-degree polynomials cannot precisely approximate ReLU, polynomial networks have to be trained from scratch and suffer from poor prediction accuracy. On the other hand, using a higher degree polynomial approximation of ReLU leads to training instability due to exploding gradients. In either case, low-degree approaches achieve lower accuracy than ReLU networks, *e.g.* on CIFAR-10 HEMET [41] and SAFENet [42] report 83.7% and 88.9% Top-1 accuracy, respectively. To mitigate gradient explosion, AESPA [46] normalized the outputs of each polynomial basis separately, leading to improved predictive accuracy.

(2) A few approaches use high-degree polynomials to approximate ReLU precisely. So, high-degree approaches do not need to train polynomial networks from scratch and can inherit weights from pretrained ReLU networks. One representative polynomial approximation of ReLU is Minimax composite polynomials [32,36]. By expressing ReLU as $\mathrm{ReLU}(x) = x \cdot (0.5 + 0.5 \cdot \mathrm{sgn}(x))$, a composite polynomial is used to approximate $\mathrm{sgn}(x)$. The approximation of ReLU is defined as $\mathrm{AppReLU}(x) = x \cdot (0.5 + 0.5 \cdot p_\alpha(x)), x \in [-1, 1]$. $p_\alpha(x)$ is the composite Minimax polynomial, and α quantifies approximation precision, *i.e.*, $|p_\alpha(x) - \mathrm{sgn}(\mathrm{x})| \leq 2^{-\alpha}$. Given $x \in [-B, B]$, the scaled AppReLU is defined as $B \cdot \mathrm{AppReLU}(x/B)$, with a precision of $B \cdot 2^{-\alpha}$. However, high-degree polynomials consume many multiplicative levels and require numerous bootstrapping operations, leading to a high computational burden. MPCNN [33], the state-of-the-art approach for secure inference of CNNs under RNS-CKKS, adopts Minimax composite polynomials with a precision of $\alpha = 13$. By choosing to approximate each ReLU function with high precision, MPCNN results in prediction accuracy that is comparable to ReLU networks. However, the same high-degree AppReLU replaces all ReLUs and consumes $\sim$ **50%** levels. The ciphertext quickly exhausts levels and uses bootstrapping to refresh the zero-level ciphertext before every AppReLU layer (refer to Figure 2). As such, the homomorphic evaluation architecture needs to be customized for each CNN architecture. Moreover, bootstrapping operations consume $>$ **70%** of inference time (refer to Table 5) and result in prohibitively high latency.

Motivation: We propose AutoFHE to address the above-mentioned limitations of existing methods for secure inference of CNNs. It is based on layerwise mixed-degree polynomials and a hybrid of approximation and training methods. The goal is to **Auto**matically generate polynomial CNNs and associated homomorphic evaluation architecture spanning a trade-off front of accuracy and latency under **FHE**.

(1) LAYERWISE MIXED-DEGREE POLYNOMIALS: A plausible solution to decrease the computational burden of secure inference is to assign layerwise mixed-degree polynomials to different ReLUs across a network based on the observation that different layers in a network have varying degree of sensitivity to approximation errors. SAFENet [42] demonstrated that mixed-degree polynomials can exploit layerwise sensitivity. However, current mixed-degree search frameworks are limited in two aspects. i) *small search space*: *e.g.* SAFENet only provides two polynomials, degree 2 and degree 3. The small search space cannot include all possible solutions from low-degree to high-degree polynomials. ii) *scalarization of multiple objectives*: a weighted sum is used to balance multiple objectives [42,45]. However, this approach requires a pre-defined preference to weigh the different objectives. So, they cannot generate diverse solutions to meet different requirements in a single optimization run.

(2) HYBRID OF APPROXIMATION AND TRAINING: Precisely approximating ReLU allows MPCNN to achieve the state-of-the-art ciphertext inference accuracy. Low-degree approaches train polynomial CNNs from scratch to compensate for loss in accuracy. We posit that by taking advantage of approximation and training, we can inherit weights from ReLU networks and fine-tune polynomial networks to adapt learnable weights to layerwise mixed-degree polynomials. In principle, such a design can allow for high ciphertext inference accuracy while reducing latency.

However, realizing the above goals presents multiple challenges. The design space, which includes layerwise mixed-degree polynomials and the associated homomorphic evaluation architectures in terms of placement of bootstrapping operations, is prohibitively large for effective manual design. Therefore, in this paper, we advocate for automated optimization of the joint polynomial and homomorphic architecture.

Contributions: We outline our contributions from two perspectives, *design* and *system*.

From a *design* perspective, our contributions are,

1. FLEXIBILITY: We jointly search for polynomial approximation and a compatible homomorphic evaluation architecture (placement of bootstrapping operations) so we can automatically adapt *any* convolutional neural networks for secure evaluation over FHE.

2. OPTIMAL APPROXIMATION: We approximate the end-to-end function represented by the CNN instead of a standalone non-linear activation function. This allows us to exploit the varying sensitivity of different layers to approximation error, and obtain polynomial approximations with high accuracy and efficient evaluation over FHE.

3. SEAMLESS DESIGN: We allow for layerwise mixed-degree polynomials, which enables us to find a range of models that span the accuracy and latency trade-off.

From a *system* perspective, our contributions are,

1. SEARCH SPACE: We design search space to include all possible low-degree and high-degree polynomials to enable us to discover better solutions.

2. SEARCH OBJECTIVE: We formulate the search problem as a multi-objective optimization. We automatically generate diverse polynomial networks spanning the trade-off front between accuracy and latency in a single optimization run.

3. SEARCH ALGORITHMS: We propose combining search and training algorithms to search over large search spaces efficiently, optimize coefficients of arbitrary polynomials, and fine-tune layerwise mixed-degree polynomial networks. Specifically, we propose:

   - *MOS*, a multi-objective search algorithm to search for solutions within a large search space.
   - *R-CCDE*, a gradient-free search algorithm to optimize coefficients of composite polynomials.
   - *PAT*, a fine-tuning algorithm that adapts network weights to polynomial activations.

Experimental Results on encrypted CIFAR datasets show that AutoFHE has a better trade-off of accuracy and latency compared to high-degree and low-degree approaches under RNS-CKKS. Compared to high-degree MPCNN [33], AutoFHE accelerates inference by $\mathbf{1.32\times \sim 1.8\times}$ while improving accuracy by $\mathbf{+0.08\%} \sim \mathbf{0.3\%}$ on CIFAR10. AutoFHE speeds up inference by $\mathbf{1.1\times \sim 1.4\times}$ while increasing accuracy by $\mathbf{+0.36\%} \sim \mathbf{0.75\%}$ on CIFAR100. Compared to low-degree AESPA [46], AutoFHE improves ciphertext accuracy by $\mathbf{+2.56\%}$ and $\mathbf{+2.44\%}$ based on ResNet32 and ResNet44 backbones on CIFAR10 with similar latency. Furthermore, we compare the accuracy-latency trade-off of networks across two FHE schemes, namely RNS-CKKS and TFHE. Specifically, we compare AutoFHE networks designed for RNS-CKKS with REDsec [18], which is designed for TFHE. We observe that AutoFHE improves accuracy by $\mathbf{+10.06\%}$ and $\mathbf{+3.46\%}$ compared to REDsec BNet$_S$ and BNet, respectively, while simultaneously reducing the corresponding latency by $\mathbf{24\times}$ and $\mathbf{103\times}$, respectively.

Code: `https://github.com/human-analysis/AutoFHE`.

2 Preliminaries

RNS-CKKS: The full residue number system (RNS) variant of Cheon-Kim-Kim-Song (RNS-CKKS) [8, 10] is a leveled homomorphic encryption (LHE) scheme for approximate arithmetic. Under RNS-CKKS, a ciphertext $\boldsymbol{c} \in \mathcal{R}^2_{Q_\ell}$ satisfies the decryption circuit $[\langle \boldsymbol{c}, sk\rangle]_{Q_\ell} = m + e$, where $\langle \cdot, \cdot \rangle$ is the dot product and $[\cdot]_Q$ is the modular reduction function. $\mathcal{R}_{Q_\ell} = \mathbb{Z}_{Q_\ell}[X]/(X^N+1)$ is the residue cyclotomic polynomial ring. The modulus is $Q_\ell = \prod_{i=0}^{\ell} q_\ell$, where $0 \le \ell \le L$. ℓ is a non-negative integer, referred to as *level*, which denotes the capacity of homomorphic multiplications. sk is the secret key with Hamming weight h. m is the original plaintext message, and e is a small error that provides security. A ciphertext has $N/2$ slots to accommodate $N/2$ complex or real numbers. RNS-CKKS supports homomorphic addition and multiplication:

$$\begin{aligned} \mathrm{Decrypt}(\boldsymbol{c} \oplus \boldsymbol{c}') &= \mathrm{Decrypt}(\boldsymbol{c}) + \mathrm{Decrypt}(\boldsymbol{c}') \approx m + m' \\ \mathrm{Decrypt}(\boldsymbol{c} \otimes \boldsymbol{c}') &= \mathrm{Decrypt}(\boldsymbol{c}) \times \mathrm{Decrypt}(\boldsymbol{c}') \approx m \times m' \end{aligned} \quad (1)$$

Bootstrapping: LHE only allows a finite number of homomorphic multiplications, with each multiplication consuming one level due to rescaling. Once a ciphertext's level reaches zero, a bootstrapping operation is required to refresh it to a higher level and allow more multiplications. The number of levels needed to evaluate a circuit is known as its *depth*. RNS-CKKS with bootstrapping [7] is an FHE scheme that can evaluate circuits of arbitrary depth. It enables us to homomorphically evaluate deep CNNs on encrypted data. Conceptually, bootstrapping homomorphically evaluates the decryption circuit and raises the modulus from Q_0 to Q_L by using the isomorphism $\mathcal{R}_{q_0} \cong \mathcal{R}_{q_0} \times \mathcal{R}_{q_1} \times \cdots \times \mathcal{R}_{q_L}$ [4]. Practically, bootstrapping [7] homomorphically evaluates modular reduction $[\cdot]_Q$ by first approximating it by a scaled sine function, which is further approximated through polynomials [7, 35]. Bootstrapping [4] has four stages: ModRaise, CoeffToSlot, EvalMod, and SlotToCoeff. Bootstrapping incurs a lot of key switching operations (KSO), which are the most time-consuming operation on the RNS-CKKS scheme [33]. The refreshed ciphertext has level $\ell = L - K$, where K levels are consumed by bootstrapping [4] for polynomial approximation of modular reduction.

Threat Model: In this paper, we assume the same threat model as prior works under HE, like HEMET [41], MPCNN [33], REDsec [18]. As discussed in the MLaaS scenario, a customer uploads encrypted data to a Cloud server and requests ML services. The Cloud uses neural networks to process the ciphertext without decryption and send back an encrypted result. Only the customer holds the secret key and can decrypt the encrypted result. The Cloud cannot learn sensitive information from the customer's data and the result. The customer is also not exposed to the Cloud's neural networks, including architectures and weights.

3 AutoFHE

Given a neural network $g(\boldsymbol{a}, \boldsymbol{\omega}_0(\boldsymbol{a}))$ with architecture $\boldsymbol{a}$ and *pretrained*[1] weights $\boldsymbol{\omega}_0(\boldsymbol{a})$ and M ReLU layers, AutoFHE generates polynomial networks on a trade-off front by maximizing accuracy and minimizing latency. For a given architecture $\boldsymbol{a}$ during the search, every solution is represented by a triplet of variables $\boldsymbol{S}(\boldsymbol{a}) = (\boldsymbol{D}(\boldsymbol{a}), \boldsymbol{\Lambda}(\boldsymbol{a}), \boldsymbol{\omega}(\boldsymbol{a}))$. We will drop the architecture $\boldsymbol{a}$ from hereon for ease of notation. $\boldsymbol{D}$ is the degree vector of all EvoReLU layers, $\boldsymbol{\Lambda}$ is the coefficients of all EvoReLU layers, and $\boldsymbol{\omega}$ are the trainable weights of the neural network which are initialized with $\boldsymbol{\omega}_0$ and fine-tuned for adaptation to the layerwise mix-degree polynomials. We will assign each solution with the minimization objective $\boldsymbol{o}(\boldsymbol{S}) = \{1 - \text{Acc}(\boldsymbol{S}), \text{Boot}(\boldsymbol{S})\}$. $1 - \text{Acc}(\boldsymbol{S})$ is the validation error, and $\text{Boot}(\boldsymbol{S})$ is the number of bootstrapping operations. Note that the accuracy and the number of bootstrapping operations depend on all the variables $\boldsymbol{S}$. For instance, appropriately adapting the network weights $\boldsymbol{\omega}$ could allow us to use lower-degree polynomial approximations of the activation functions, which reduces the required number of bootstrapping calls.

AutoFHE is a search-based approach which comprises of three main components: *search space* (Section 3.1), *search objective* (Section 3.2) and *search algorithm* (Section 3.3). We describe each of these in detail next.

3.1 Search Space

EvoReLU is a genetic polynomial function used to replace the non-arithmetic ReLU function. We model it as follows,

$$y = \text{EvoReLU}(x) = \begin{cases} x, & d = 1 \\ \alpha_2 x^2 + \alpha_1 x + \alpha_0, & d = 2 \\ x \cdot (\mathcal{F}(x) + 0.5), & d > 2 \end{cases} \tag{2}$$

where $\mathcal{F}(x)$ is a composite polynomial with K sub-polynomials $f_k^{d_k}$ with degree d_k and seeks to approximate $0.5 \cdot \text{sgn}(x)$ for $d > 2$.

$$\mathcal{F}(x) = (f_K^{d_K} \circ \cdots \circ f_k^{d_k} \circ \cdots \circ f_1^{d_1})(x), 1 \leq k \leq K \tag{3}$$

The total degree of $\mathcal{F}(x)$ is $\prod_{k=1}^{K} d_k$, and consequently the degree of EvoReLU is $d = \prod_{k=1}^{K} d_k + 1$. For $d > 2$ the **multiplicative depth** of EvoReLU is $1 + \sum_{k=1}^{K} \lceil \log_2(d_k + 1) \rceil$ when using the Baby-Step Giant-Step (BSGS) algorithm [4, 35] to evaluate composite polynomial $\mathcal{F}(x)$. For $d = 1$ and $d = 2$, the multiplicative depth is 0 and 2, respectively.

EvoReLU is modeled to allow for *automatic* discovery of common solutions in the literature for improving the latency of inference on ciphertexts. For instance, for $d = 1$, $\text{EvoReLU}(x) = x$ is equivalent to removing the corresponding ReLU layer through pruning [27, 42, 45]. Similarly, for $d = 2$, $\text{EvoReLU}(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$ is equivalent to using low-degree approximations of ReLU through quadratic functions [6, 12, 20, 41, 42, 45]. Finally, for $d > 2$, $\text{EvoReLU}(x) = x \cdot (\mathcal{F}(x) + 0.5)$ is equivalent to high-degree polynomial approximations [32, 33] of ReLU. In this case, EvoReLU bears similarity to the Minimax composite polynomial in Lee *et al.* [33, 36]. However, the objective for optimizing the coefficients differs significantly. While Lee *et al.* [33, 36] seek to approximate a single ReLU function precisely, our goal is to jointly optimize all EvoReLU functions in a neural network $\boldsymbol{a}$ to approximate its corresponding function $g(\boldsymbol{a}, \boldsymbol{\omega})$.

For the quadratic version of EvoReLU, we allow the coefficients α_0, α_1, and α_2 to differ *channel-wise* when fine-tuning polynomial CNNs. Such a design improves performance and makes optimizing model weights $\boldsymbol{\omega}$ more stable. For this case, we also introduce a BatchNorm layer after the quadratic EvoReLU for recentering and reshaping the distribution of output activations. We can integrate the BatchNorm parameters into the polynomial coefficients. So, there is no extra consumption of multiplicative levels.

We **represent** the composite polynomial $\mathcal{F}(x)$ by its degree vector $\boldsymbol{d} = \{d_k\}_{k=1}^{K}, d_k \in \mathbb{N}$. Each sub-polynomial $f_k^{d_k}(x)$ as a linear combination of Chebyshev polynomials of degree d_k,

$$f_k^{d_k}(x) = \frac{1}{\beta_k} \sum_{i=1}^{d_k} \alpha_i \text{T}_i(x) \tag{4}$$

where $\alpha_i \in \mathbb{R}$ and $\beta_k \in \mathbb{R}$. $\text{T}_i(x)$ is the Chebyshev basis of the first kind, α_i are the coefficients for linear combination, and scaling parameter β_k is a parameter to scale the output. The coefficients $\boldsymbol{\alpha}_k = \{\alpha_i\}_{i=1}^{d_k}$ control the polynomial's shape, while β_k controls its amplitude. A composite polynomial with the degree vector $\boldsymbol{d}$ has learnable parameters:

$$\boldsymbol{\lambda} = \{\boldsymbol{\alpha}_1, \beta_1, \cdots, \boldsymbol{\alpha}_k, \beta_k, \cdots, \boldsymbol{\alpha}_K, \beta_K\} \tag{5}$$

A neural network with M ReLU activations needs M EvoReLU polynomial activations. $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_M\}$ is

[1] In our paper, *pretrained* neural networks especially refer to neural networks with ReLU activation.

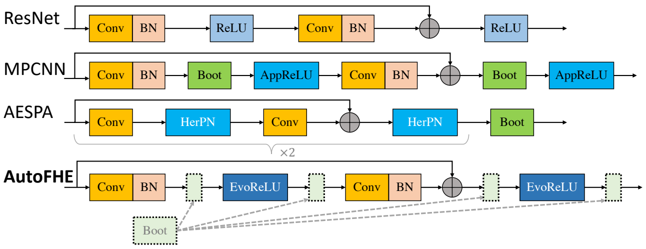

Figure 2: Homomorphic evaluation architectures for Residual Networks (ResNets). 1st row: standard Conv-BN-ReLU triplet [24]. 2nd row: MPCNN [33] which uses high-degree polynomials. 3rd row: AESPA [46] which uses low-degree polynomials. 4th row: The proposed AutoFHE, which uses layerwise mixed-degree polynomials. Dashed rectangles indicate the plausible locations where bootstrapping can be placed. Both EvoReLU and the placement of bootstrapping operations are searched.

the degree vector of all EvoReLUs and the corresponding coefficient parameters are $\boldsymbol{\Lambda} = \{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \cdots, \boldsymbol{\lambda}_M\}$.

Homomorphic Evaluation Architecture refers to the placement of ConvBN, polynomial, and bootstrapping. In MPCNN, AppReLU depth is 14, ConvBN depth is 2, and the remaining levels after bootstrapping are $L - K = 16$. So, bootstrapping is placed after every AppReLU and ConvBN (Figure 2). AESPA [46] uses degree 2 Hermite polynomial (HerPN) with depth 2 to replace ReLU and Batchnorm. Therefore, every 4 Conv-HerPN should be followed by one bootstrapping operation (Figure 2). In AutoFHE, EvoReLU is layerwise and mixed-degree. The multiplicative depth of EvoReLU varies layer by layer. AutoFHE introduces a flexible evaluation architecture where bootstrapping can be called after ConvBN or EvoReLU (Figure 2). The flexible evaluation architecture of AutoFHE can better fit layerwise mix-degree EvoReLU.

Linear Scaling: Since the domain of Chebyshev polynomials and bootstrapping is $[-1, 1]$, we need to scale the ciphertext to $[-1, 1]$ before Chebyshev polynomials and bootstrapping and reverse it after Chebyshev polynomials and bootstrapping. To avoid consuming levels, scaling is integrated into other operations. In MPCNN, the polynomial neural network with AppReLU can be roughly regarded as *piece-wise linear* due to high precision approximation of high-degree AppReLU. MPCNN estimates domain of AppReLU using the training dataset and uses the maximum number of all AppReLU domains to scale down input images by $\times 1/B$ and scale up ciphertext in the fully connected layer by $\times B$. In AESPA, HerPN uses degree 2 Hermite polynomial and does not need to scale input of HerPN. However, the input of bootstrapping should be scaled to $[-1, 1]$. The domain $[-B, B]$ of each bootstrapping can be estimated on the training dataset, then integrate $\times 1/B$ into the previous HerPN and integrate $\times B$ into the next Conv layer. In AutoFHE, bootstrapping can be placed before EvoReLU and after EvoReLU. So, we need to estimate domain $[-B_{in}, B_{in}]$ and range $[-B_{out}, B_{out}]$ of each EvoReLU on the training dataset. Figure 3 shows possible scaling scenarios in AutoFHE. We can integrate scaling into the previous or next operation. Specifically, we can multiply convolutional weight, batch norm weight and bias, and polynomial coefficients by the scaling constant. The scaling operation is conducted in plaintext and will not introduce either extra level consumption. With the scaling EvoReLU can now be expressed as,

$$\text{EvoReLU}(x) = \begin{cases} x, & d = 1 \\ \alpha_2 x^2 + \alpha_1 x + \alpha_0, & d = 2 \\ x \cdot \left(\mathcal{F}(x/B) + 0.5\right), & d > 2 \end{cases} \tag{6}$$

Search Space: Our search space includes the number of sub-polynomials (K) in our composite polynomial, the choice of degrees for each sub-polynomial (d_k), and the coefficients of

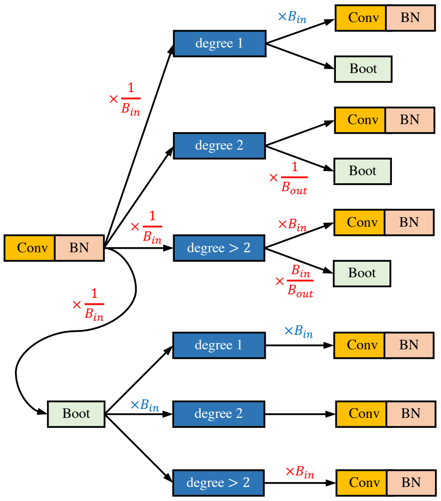

Figure 3: Scaling in AutoFHE. Color Key: integrate scaling into previous operation; integrate scaling into next operation.

Variable	Option
# polynomials (K)	6
poly degree (d_k)	$\{0,1,3,5,7\}$
coefficients $(\boldsymbol{\Lambda})$	$\mathbb{R}$

Table 1: Search variables and options.

Backbone	#ReLUs	Dimension of $\boldsymbol{D}$	Search Space Size
ResNet20	19	114	10^{80}
ResNet32	31	186	10^{130}
ResNet44	43	258	10^{180}
VGG11	10	60	10^{42}

Table 2: Search space of AutoFHE for ResNet and VGG.

the polynomials $\mathbf{\Lambda}$. Table 1 shows each variable's options. Note that choice $d_k = 0$ corresponds to an identity placeholder, so theoretically, the composite polynomial may have fewer than K sub-polynomials. Furthermore, when the degree of $(p_k^{d_k} \circ p_{k-1}^{d_{k-1}})(x)$ is less than or equal to 31 (maximum degree of a polynomial supported on RNS-CKKS [32, 36]), we merge the two sub-polynomials into a single sub-polynomial $p_k^{d_k}(p_{k-1}^{d_{k-1}})(x)$ with degree $d_k \cdot d_{k-1} \leq 31$ before computing its depth. This helps reduce the size of the search space and leads to smoother exploration. Table 2 lists the number of ReLUs of our backbone models and the corresponding dimension and size of search space for $\boldsymbol{D}$. Searching for layerwise EvoReLU is a challenging high-dimensional optimization problem within a vast search space.

3.2 Search Objective

AutoFHE formulates the search problem as *a multi-objective* optimization

$$\begin{aligned} \min_{\boldsymbol{D}} \quad & \{1 - \text{Acc}_{val}\left(g\left(\boldsymbol{\omega}^* \mid \boldsymbol{D}, \mathbf{\Lambda}(\boldsymbol{D})\right)\right), \text{Boot}(\boldsymbol{D})\} \\ \text{s.t.} \quad & \boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \mathcal{L}_{train}\left(g\left(\boldsymbol{\omega} \mid \boldsymbol{D}, \mathbf{\Lambda}(\boldsymbol{D})\right)\right) \\ & \mathcal{L}_{train} = (1-\tau)\mathcal{L}_{CE} + \tau\mathcal{L}_{KL} \end{aligned} \tag{7}$$

where $g(\boldsymbol{\omega})$ is a neural network with M activation layers and the trainable network weight $\boldsymbol{\omega}$. The outer multi-objective minimization formulation $\min_{\boldsymbol{D}} \{1 - \text{Acc}_{val}\left(g\left(\boldsymbol{\omega}^* \mid \boldsymbol{D}, \mathbf{\Lambda}(\boldsymbol{D})\right)\right), \text{Boot}(\boldsymbol{D})\}$ for $\boldsymbol{D}$ is to *maximize* the validation accuracy Acc_{val} as well as *minimize* the number of bootstrapping operations. The coefficient vector $\mathbf{\Lambda}$ is formulated as a function of $\boldsymbol{D}$. In Equation 7, Acc_{val} is the Top-1 accuracy on a validation dataset *val*, Boot is the number of bootstrapping operations. To determine the number of bootstrapping operations, we count the level consumption of all EvoReLU's to determine where we need to call bootstrapping. By minimizing the number of bootstrapping operations, we search for the placement of bootstrapping and minimize the wasted levels. For example, consider that we have a ciphertext with a level equal to 2, but the next operation consumes 10 levels. We must waste 2 levels and call bootstrapping to refresh the ciphertext first. AutoFHE can minimize the wasted levels by adjusting the depth of EvoReLU. $\{\boldsymbol{D}_i, \mathbf{\Lambda}_i\}$ has its corresponding network weight $\boldsymbol{\omega}_i$ that can compensate errors introduced by layerwise EvoReLU $\{\boldsymbol{D}_i, \mathbf{\Lambda}_i\}$. We initialize $\boldsymbol{\omega}_i$ with the weight $\boldsymbol{\omega}_0$ from the pretrained ReLU network and then fine-tune the network $g(\boldsymbol{\omega}_i)$ to minimize the training loss $\mathcal{L}_{train}(\boldsymbol{\omega}_i)$ on the training dataset. In summary, the objective in Equation 7 guides the search algorithm to i) explore layerwise EvoReLU, including its *degrees* and *coefficients*; 2) discover the placement of bootstrapping to work well with layerwise mixed-degree EvoReLU; 3) trade-off validation accuracy and inference latency to return diverse polynomial networks.

The training loss $\mathcal{L}_{train}$ used to optimize the weight $\boldsymbol{\omega}$ is $(1-\tau)\mathcal{L}_{CE} + \tau\mathcal{L}_{KL}$, where $\mathcal{L}_{CE}$ is the cross-entropy loss and $\mathcal{L}_{KL}$ is the Kullback–Leibler (KL) divergence loss. τ is a predefined parameter to balance CE and KL loss. In Equation 7, we omit the variable of $\mathcal{L}_{train}$. Given EvoReLU $(\boldsymbol{D}, \mathbf{\Lambda})$, the variable is weight $\boldsymbol{\omega}$. The KL loss computes the distance between distributions of logits of the polynomial network and the ReLU network. We introduce the KL loss because it can push the output of the polynomial network close to the ReLU network. It can be regarded as knowledge distillation (KD) [25]. We transfer knowledge from the ReLU network to polynomial networks.

3.3 Search Algorithms

3.3.1 Multi-Objective Search

Multi-Objective Optimization: Given two solutions with minimization objectives $\boldsymbol{o}_1, \boldsymbol{o}_2 \in \mathbb{R}^d$, we want to minimize all items of $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$. If $\boldsymbol{o}_{1,i} \leq \boldsymbol{o}_{2,i}, \forall i \in \{1, 2, \cdots, d\}$ and $\boldsymbol{o}_{1,j} < \boldsymbol{o}_{2,j}, \exists j \in \{1, 2, \cdots, d\}$, $\boldsymbol{o}_1$ *dominates* $\boldsymbol{o}_2$ [13, 52]. It means $\boldsymbol{o}_1$ is better than $\boldsymbol{o}_2$. It is denoted as $\boldsymbol{o}_1 \prec \boldsymbol{o}_2$. A set of solutions $\boldsymbol{O} = \{\boldsymbol{o}_i\}_{i=1}^N$ can be grouped into sub-sets, $\{\boldsymbol{o}_i\}_{i=1}^{N_1}$, $\{\boldsymbol{o}_i\}_{i=1}^{N_2}$, $\cdots$, corresponding to the 1st trade-off front, the 2nd trade-off front and so on. Solutions within the same trade-off front are not dominated by each other. The 1st trade-off front dominates the 2nd trade-off front, and so on.

MOS: Figure 4 and Algorithm 1 show our **M**ulti-**O**bjective **S**earch framework for AutoFHE. MOS is an evolutionary search algorithm to solve the multi-objective optimization in Equation 7. It maintains a population of solutions distributed on trade-off fronts of accuracy and the number of bootstrapping operations. We crossover and mutate solutions to improve every generation's trade-off fronts, as shown in Figure 4. During the search, we define the population size, namely the number of solutions, N. These N solutions may be grouped into multiple trade-off fronts, which can improve exploration ability during search. We design operations to search for layerwise mixed-degree polynomials:

① SELECT: Solutions of the current population are first grouped to different trade-off fronts by non-dominated sorting [13]. Solutions on the same trade-off front have the same fitness. The 1st trade-off front has a higher fitness than the 2nd trade-off front, and so on. We apply the tournament selection [21] to improve the diversity of offspring. Specifically, we randomly select three solutions from the current population and keep the solution with the highest fitness. We choose N' solutions from the current population to build the offspring. In our paper, we set $N' = 6N$.

② CROSSOVER enables network-level information exchange. As shown in Figure 4, we randomly and uniformly exchange EvoReLU layers between two solutions

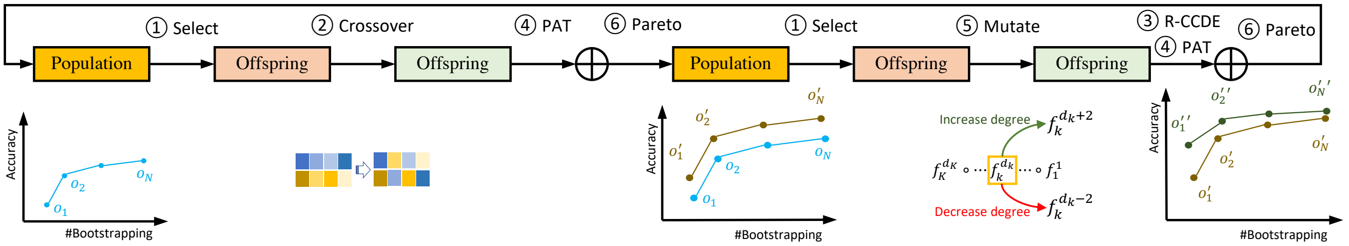

Figure 4: Multi-objective search of AutoFHE.

```
Algorithm 1: MOS
Input  : Pretrained Network $g(\boldsymbol{\omega}_0)$, population size $N$,
         offspring size $N'$, number of generations $T$,
         training dataset Train, mini-validation dataset
         Minival;
Output : Trade-off front $\{\boldsymbol{D}_i,\boldsymbol{\Lambda}_i,\boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \le i \le N$ ;
for $t \leftarrow 1$ to $T$ do
    $\{\boldsymbol{D}_j,\boldsymbol{\Lambda}_j\}_{j=1}^{N'} \leftarrow$ Select($\{\boldsymbol{D}_i,\boldsymbol{\Lambda}_i,\boldsymbol{o}_i\}_{i=1}^{N}$) ;
    $\{\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j\}_{j=1}^{N'} \leftarrow$ Crossover($\{\boldsymbol{D}_j,\boldsymbol{\Lambda}_j\}_{j=1}^{N'}$) ;
    $\{\boldsymbol{\omega}'_j\}_{j=1}^{N'} \leftarrow$ PAT($\{\boldsymbol{\omega}_0;\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j\}_{j=1}^{N'}$, Train) ;
    $\{\boldsymbol{o}'_j\}_{j=1}^{N'} \leftarrow$ Eval($\{\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j,\boldsymbol{\omega}'_j\}_{j=1}^{N'}$, Minival) ;
    $\left\{\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j,\boldsymbol{\omega}'_j\right\} \Rightarrow \boldsymbol{o}'_j, 1 \le j \le N'$ ;
    $\{\boldsymbol{o}_i\}_{i=1}^{N} \leftarrow$ Pareto($\{\boldsymbol{o}_i\}_{i=1}^{N} \cup \{\boldsymbol{o}'_j\}_{j=1}^{N'}$) ;
    $\{\boldsymbol{D}_i,\boldsymbol{\Lambda}_i,\boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \le i \le N$ ;
    $\{\boldsymbol{D}_j,\boldsymbol{\Lambda}_j,\boldsymbol{\omega}_j\}_{j=1}^{N'} \leftarrow$ Select($\{\boldsymbol{D}_i,\boldsymbol{\Lambda}_i,\boldsymbol{\omega}_i,\boldsymbol{o}_i\}_{i=1}^{N}$) ;
    $\{\boldsymbol{D}'_j\}_{j=1}^{N'} \leftarrow$ Mutate($\{\boldsymbol{D}_j\}_{j=1}^{N'}$) ;
    $\{\boldsymbol{\Lambda}'_j\}_{j=1}^{N'} \leftarrow$ R-CCDE($\{\boldsymbol{D}'_j\}_{j=1}^{N'}$) ;
    $\{\boldsymbol{\omega}'_j\}_{j=1}^{N'} \leftarrow$ PAT($\{\boldsymbol{\omega}'_j;\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j\}_{j=1}^{N'}$, Train) ;
    $\{\boldsymbol{o}'_j\}_{j=1}^{N'} \leftarrow$ Eval($\{\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j,\boldsymbol{\omega}'_j\}_{j=1}^{N'}$, Minival) ;
    $\left\{\boldsymbol{D}'_j,\boldsymbol{\Lambda}'_j,\boldsymbol{\omega}'_j\right\} \Rightarrow \boldsymbol{o}'_j, 1 \le j \le N'$ ;
    $\{\boldsymbol{o}_i\}_{i=1}^{N} \leftarrow$ Pareto($\{\boldsymbol{o}_i\}_{i=1}^{N} \cup \{\boldsymbol{o}'_j\}_{j=1}^{N'}$) ;
    $\{\boldsymbol{D}_i,\boldsymbol{\Lambda}_i,\boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \le i \le N$ ;
```

(parents) to generate two new solutions. However, new solutions cannot inherit weights from parent solutions because weights are adapted to the parent's layerwise mixed-degree polynomials. So, we initialize weights with pretrained weights and then fine-tune new solutions.

③ R-CCDE optimizes EvoReLU coefficients.

④ PAT fine-tunes new polynomial CNNs.

⑤ MUTATION is to locally explore EvoReLU. We randomly increase or decrease the degree to smoothly change polynomials, as shown in Figure 4. We randomly increase or decrease the degree of a sub-polynomial of EvoReLU with predefined probabilities.

⑥ PARETO refers to non-dominated sorting and crowding distance sorting [13]. We apply Pareto to select N solutions from both population and offspring ($N+N'$ solutions) to build a new population (N solutions).

3.3.2 R-CCDE

Coevolution: The composite polynomial used by EvoReLU is: $y_1 = f_1^{d_1}(x|\alpha_1,\beta_1),\cdots,y_K = f_K^{d_K}(y_{K-1}|\alpha_K,\beta_K)$. The forward architecture of the composite polynomial, $x \mapsto y_1 \mapsto y_2 \cdots \mapsto y_{K-1} \mapsto y$ is suitable for *coevolution* [43, 44, 58] and provides a natural *decomposition*. We can sequentially adjust every sub-polynomial to push the output y close to the target non-arithmetic function. Given the degree $\boldsymbol{d}$, the learnable parameter of EvoReLU $\boldsymbol{\lambda} = (\alpha_1,\beta_1,\cdots,\alpha_K,\beta_K)$ is grouped into $\{\alpha_1\},\{\beta_1\},\cdots,\{\alpha_K\},\{\beta_K\}$. The coefficient α controls the shape of the sub-polynomial output, while the scaling parameter β controls the amplitude. We sequentially update $\{\alpha_k\}$ followed by β_k, $1 \le k \le K$, since i) sub-polynomials close to input will have a larger effect on the output, and ii) it is easier to learn coefficients by decoupling the amplitude from the coefficients.

Differentiable Evolution: The EvoReLU variables $\{\alpha_k\}_{k=1}^{K}$ and $\{\beta_k\}_{k=1}^{K}$ are in the continuous space. We adopt a simple yet effective search algorithm to optimize these variables. Differentiable evolution (DE) [48] only uses the *difference* between solutions to optimize continuous variables. Given the following minimization problem in the continuous space

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} \mathcal{F}(\boldsymbol{x}) \tag{8}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ and $\mathcal{F}$ is the minimization objective. DE maintains a set of solutions $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}, \boldsymbol{x}_i \in \mathbb{R}^d$. The mutation, crossover, and selection of DE are defined as:

$$\begin{aligned}
&\text{Mutation: } \boldsymbol{v} = \boldsymbol{x}_i + F\cdot(\boldsymbol{x}_j - \boldsymbol{x}_k), 1 \le i,j,k \le N \\
&\text{Crossover: } \boldsymbol{u}[t] = \begin{cases} \boldsymbol{v}[t], & \mathcal{U}(0,1) \le CR \\ \boldsymbol{x}_i[t], & \text{Otherwise} \end{cases}, 1 \le t \le d \\
&\text{Selection: } \boldsymbol{u} = \begin{cases} \boldsymbol{u}, & \mathcal{F}(\boldsymbol{u}) \le \mathcal{F}(\boldsymbol{x}_i) \\ \boldsymbol{x}_i, & \text{Otherwise} \end{cases}
\end{aligned} \tag{9}$$

Algorithm 2: R-CCDE

```
Input   : Composite polynomial $\mathcal{F}(x) = (f_K^{d_K} \circ f_{k-1}^{d_{k-1}} \circ \cdots \circ f_1^{d_1})(x)$ with parameters $\boldsymbol{\lambda} = \{\boldsymbol{\alpha}_1, \beta_1, \cdots, \boldsymbol{\alpha}_k, \beta_k, \cdots \boldsymbol{\alpha}_K, \beta_K\}$,
          target function $q(x)$, number of generations $T$, scaling decay $\gamma$ ;
Output  : Context vector $\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_k^*, \beta_k^*, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
Initial : $\boldsymbol{\lambda}^* \leftarrow \texttt{LHS}(\sum_{k=1}^{K} d_k + K)$
for $t \leftarrow 1$ to $T$ do
    for $k \leftarrow 1$ to $K$ do
        $\boldsymbol{\alpha}_k^\star \leftarrow \arg\min_{\boldsymbol{\alpha}_k} \mathcal{L}_{\mathcal{F},q}(\boldsymbol{\alpha}_k|\boldsymbol{\lambda}^*)$   s.t.   $\boldsymbol{\alpha}_k|\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
        $\boldsymbol{\lambda}^* \leftarrow (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_k^\star, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
        $\beta_k^\star \leftarrow \arg\min_{\beta_k} \mathcal{L}_{\mathcal{F},q}(\beta_k|\boldsymbol{\lambda}^*) + \gamma \cdot {\beta_k}^2$   s.t.   $\beta_k|\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \beta_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
        $\boldsymbol{\lambda}^* \leftarrow (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \beta_k^\star, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
```

where $F \in \mathbb{R}$ is the scaling factor, $CR \in \mathbb{R}$ is the crossover rate, and $\mathcal{U}(0,1)$ is the uniform distribution between 0 and 1. Equation 9 shows a simple strategy to update solutions by only using difference. First, mutation updates $\boldsymbol{x}_i$ with the scaled difference $F \cdot (\boldsymbol{x}_j - \boldsymbol{x}_k)$. Then, we randomly select items from $\boldsymbol{v}$ or $\boldsymbol{x}_i$ to generate a new solution $\boldsymbol{u}$. Finally, we evaluate $\mathcal{F}(\boldsymbol{u})$ and use $\boldsymbol{u}$ to replace $\boldsymbol{x}_i$ if $\mathcal{F}(\boldsymbol{u}) \leq \mathcal{F}(\boldsymbol{x}_i)$. DE only uses difference and does not suffer from gradient exploding. It maintains a set of solutions and is not sensitive to initialization.

R-CCDE: We propose **R**egularized **C**ooperative **C**oevolution **D**ifferentiable **E**volution, called R-CCDE, to search for parameters of EvoReLU$(x, \boldsymbol{\lambda}; \boldsymbol{d})$, namely $\boldsymbol{\lambda} = \{\boldsymbol{\alpha}_k, \beta_k\}_{k=1}^{K}$. The scaling parameters $\{\beta_k\}_{k=1}^{K}$ are used to adjust the amplitude of sub-polynomials during the search. After the search, $\{\beta_k\}_{k=1}^{K}$ will be used to scale $\{\boldsymbol{\alpha}_k\}_{k=1}^{K}$ and obtain coefficients of polynomials. The decomposition makes the search easier by decoupling the shape and amplitude of polynomials. We detail the implementation of R-CCDE in Algorithm 2. R-CCDE takes as input a composite polynomial $\mathcal{F}(x) = (f_K^{d_K} \circ f_{k-1}^{d_{k-1}} \circ \cdots \circ f_1^{d_1})(x)$ with parameters $\boldsymbol{\lambda} = \{\boldsymbol{\alpha}_1, \beta_1, \cdots \boldsymbol{\alpha}_k, \beta_k \cdots \boldsymbol{\alpha}_K, \beta_K\}$. Because EvoReLU is defined as $y = \text{EvoReLU}(x) = x \cdot (\mathcal{F}(x) + 0.5)$ in Equation 3, we use the composite polynomial $\mathcal{F}(x)$ to approximate $q(x) = 0.5 \cdot \text{sgn}(x)$. We set the number of generations and the scaling decay parameter to T and γ, respectively. The objective function $\mathcal{L}_{\mathcal{F},q}(\cdot)$ is the ℓ_1 distance between the composite polynomial $\mathcal{F}(x)$ and the target function $q(x)$. R-CCDE maintains a *context vector* [44] $\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$ as the best solution so far. $\boldsymbol{\lambda}^*$ is initialized via Latin hypercube sampling (LHS). In Algorithm 2, $\boldsymbol{\alpha}_k$ and β_k $1 \leq k \leq K$ are optimized using DE sequentially and alternatively. In generation t, given the k-th position, we optimize $\boldsymbol{\alpha}_k$ as

$$
\begin{aligned}
&\boldsymbol{\alpha}_k^\star = \arg\min_{\boldsymbol{\alpha}_k} \mathcal{L}_{\mathcal{F},q}(\boldsymbol{\alpha}_k|\boldsymbol{\lambda}^*) \\
&\text{s.t.} \quad \boldsymbol{\alpha}_k|\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots \boldsymbol{\alpha}_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)
\end{aligned}
\tag{10}
$$

where $\boldsymbol{\alpha}_k$ is a variable, while other $\boldsymbol{\alpha}$'s and β's are fixed. A candidate solution of $\boldsymbol{\alpha}_k$ is plugged into $\boldsymbol{\lambda}^*$. Then, we evaluate the candidate solution $\boldsymbol{\alpha}_k$ by evaluating $\mathcal{L}_{\mathcal{F},q}(\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$. We adopt DE to solve the single-objective optimization problem in the continuous space. We maintain a set of candidate solutions of $\boldsymbol{\alpha}_k$, namely $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}, \boldsymbol{x}_i \in \mathbb{R}^{d_k}$. Mutation, crossover, and selection defined in Equation 9 are applied to update solutions in $\boldsymbol{X}$. Then, the best solution in $\boldsymbol{X}$ is assigned to $\boldsymbol{\alpha}_k^\star$. We use $\boldsymbol{\alpha}_k^\star$ to replace $\boldsymbol{\alpha}_k^*$ in the context vector $\boldsymbol{\lambda}^*$ to update $\boldsymbol{\lambda}^*$

$$
\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots \boldsymbol{\alpha}_k^\star, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*) \tag{11}
$$

In summary, i)$\{\boldsymbol{\alpha}_k\}_{k=1}^{K}$ and $\{\beta_k\}_{k=1}^{K}$ *separately* maintain their sets of solutions that are optimized by DE; ii) the *context* vector $\boldsymbol{\lambda}^*$ is not only the best solution so far. It allows different variables to share information. When evolving $\{\beta_k\}_{k=1}^{K}$, the objective introduces a *regularization* term

$$
\begin{aligned}
&\beta_k^\star = \arg\min_{\beta_k} \underbrace{\mathcal{L}_{\mathcal{F},q}(\beta_k|\boldsymbol{\lambda}^*)}_{\ell_1 \text{ Distance}} + \underbrace{\gamma \cdot {\beta_k}^2}_{\text{Regularization}} \\
&\text{s.t.} \quad \beta_k|\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \beta_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)
\end{aligned}
\tag{12}
$$

where $\gamma \cdot \beta_k^2$ is the regularization term and γ is the scaling decay parameter. Without the regularization $\gamma \cdot \beta_k^2$, we observe $\{\beta_k\}_{k=1}^{K}$ prefers *large* numbers. Because $p_k^{d_k}(x) = \frac{1}{\beta_k} \sum_{i=1}^{d_k} \alpha_i \mathrm{T}_i(x)$, large $\{\beta_k\}_{k=1}^{K}$ numbers can make the composite polynomial *numerical stable* because the polynomial output is scaled to a small number. However, it is hard to distinguish different solutions of $\{\boldsymbol{\alpha}_k\}_{k=1}^{K}$. By introducing the regularization term $\gamma \cdot \beta_k^2$, DE prefers large numbers in earlier generations and gradually reduces β_k. Therefore, DE is not biased toward solutions with large β_k numbers. We use R-CCDE to optimize coefficients of quadratic and high-degree EvoReLU. For quadratic EvoReLU in Equation 2, α_2 is obtained by R-CCDE and $\alpha_1 = 0.5$, $\alpha_0 = 0$.

3.3.3 Polynomial-Aware Training

Replacing ReLU with EvoReLU in pretrained neural networks injects *minor* approximation errors, which leads to performance loss. Fine-tuning can mitigate this performance loss by allowing the learnable weights (e.g., convolution or fully connected layers) to adapt to the approximation error. However, backpropagation through EvoReLU easily leads to exploding gradients since the gradients may be amplified exponentially due to many composite polynomials. From Equation 2, high-degree EvoReLU can precisely approximate ReLU, while ReLU pruning (degree 1) and quadratic EvoReLU (degree 2) have a larger approximation error. For low-degree EvoReLU (degree ≤ 2), we can use SGD to compute gradients because they do not amplify gradients. For high-degree EvoReLU (degree > 2), we can use gradients from the original non-arithmetic ReLU function for *backpropagation*. Specifically, during *the forward* pass, EvoReLU injects slight errors captured by the training loss. During the *backward* pass, we bypass high-degree EvoReLU and use ReLU to compute gradients to update the weights of the linear trainable layers. We refer to this procedure as **P**olynomial-**A**ware **T**raining (PAT). PAT is inspired by STE [3] and QAT [26], which uses two different functions for forward- and back-propagation. PAT is defined as:

$$\frac{\partial \text{EvoReLU}(x)}{\partial x} = \begin{cases} 1, & d = 1 \\ 2\alpha_2 x + \alpha_1, & d = 2 \\ \partial \text{ReLU}(x)/\partial x, & d > 2 \end{cases} \tag{13}$$

4 Experiments

Datasets: We benchmark AutoFHE on CIFAR10 and CIFAR100 [31]. Both datasets have 50,000 training and 10,000 validation images at a resolution of 32×32. CIFAR10 has 10 classes, while CIFAR100 includes 100 classes. The validation images are treated as private data and used only for evaluating the final networks. To guide the search process, we randomly select 10,000 images from the training split as a *minival* [53] dataset and use the Top-1 accuracy on the minival dataset to optimize Equation 7. In addition, PAT uses the training split to fine-tune polynomial networks. Finally, as our final result, we report the Top-1 accuracy on the encrypted validation dataset under RNS-CKKS.

Method	Venue	Scheme	Polynomial	Layerwise	Strategy	Arch
MPCNN [33]	ICML22	CKKS	high-degree	×	approx	manual
AESPA [46]	arXiv22	CKKS	low-degree	×	train	manual
REDsec [18]	NDSS23	TFHE	n/a	n/a	train	manual
AutoFHE	USENIX24	CKKS	mixed	✓	adapt	search

Table 3: AutoFHE and baselines. AutoFHE, MPCNN and AESPA use fixed-point arithmetic under RNS-CKKS, while REDsec adpots ternary neural networks under TFHE.

Parameters: (1) Training Parameters: We train ReLU networks (used by MPCNN and AutoFHE) and AESPA using SGD optimizer with batch size 128, epochs 200, learning rate 0.1, momentum 0.9 and weight decay 0.0005. We use a cosine learning rate scheduler. We clip gradients to 1 when we train polynomial networks (AESPA or AutoFHE). **(2)** Search Parameters: For MOS, we set the number of generations to 10. Population size is proportional to the number of variables. We set population size to 10, 20, 30 and 40 for VGG11, ResNet20, ResNet32 and ResNet44, respectively. The offspring size is 6× as the population size. When we mutate a polynomial, its degree is decreased by -2 with a probability of 0.5 and is increased by $+2$ with a probability of 0.3. For R-CCDE, we set the search domain of $\boldsymbol{\alpha}$ to $[-5,5]$ and that of β to $[1,5]$. We use the set of 20 solutions for optimizing β. For $\boldsymbol{\alpha}$, we set the number of solutions equal to 20× the number of variables. We set both the scaling factor F and the crossover rate CR to 0.5. We set the scaling decay to $\gamma = 0.01$ and the number of iterations to 100. We run R-CCDE 10 times with different random seeds and retain the best solution. **(3)** Finetuning Parameters: To fine-tune AutoFHE polynomial networks, we train them using PAT with batch size 128, learning rate 0.02, momentum 0.9, weight decay 0.0005, and KL weight $\tau = 0.9$. We clip gradients to 1. During the search, to quickly estimate the accuracy of polynomial networks, we set epochs to 5. After the search, we set epochs to 90 and use the cosine annealing learning rate scheduler. **(4)** Cryptographic Parameters: We followed MPCNN to set the same cryptographic parameters [33] of RNS-CKKS for MPCNN, AESPA and AutoFHE. The cyclotomic polynomial uses degree $N = 2^{16}$. The Hamming weight of the secret key is 192. The ciphertext level is $L = 30$, while bootstrapping uses 14 levels ($K = 14$). Base modulus, special modulus, and bootstrapping modulus are set to 51 bits, while default modulus is set to 46 bits [33]. The cryptographic parameters satisfy *128-bit security* [9, 33].

Hardware and RNS-CKKS Library: (1) Search: On one NVIDIA RTX A6000 GPU, the search process for ResNet-20/32/44 and VGG11 on CIFAR10 took 44 hours, 64 hours, 88 hours, 13 hours, respectively. The search for ResNet32 and VGG11 on CIFAR100 took 67 and 12 hours, respectively. To accelerate R-CCDE, we use 100 CPU threads of AMD EPYC 7502 32-core processor. **(2)** FHE Inference: We evaluate latency under FHE on the publicly available Amazon AWS instance, r5.24xlarge, which has 96 CPU threads and 768 GB RAM. We build C++ implementation of AutoFHE under RNS-CKKS on top of MPCNN using Microsoft SEAL library [50]. We adopt MPCNN implementations of Conv, BN, Downsample, AvgPool, and FC layers.

Baselines: We compare the proposed AutoFHE with two recent state-of-the-art approaches under RNS-CKKS, high-degree polynomial and approximation-based approach MPCNN [33] and low-degree polynomial and training-based approach AESPA [46], as shown in Table 3. We also benchmark against REDsec [18] under TFHE to compare different FHE schemes.

RNS-CKKS Baselines: MPCNN uses high-degree

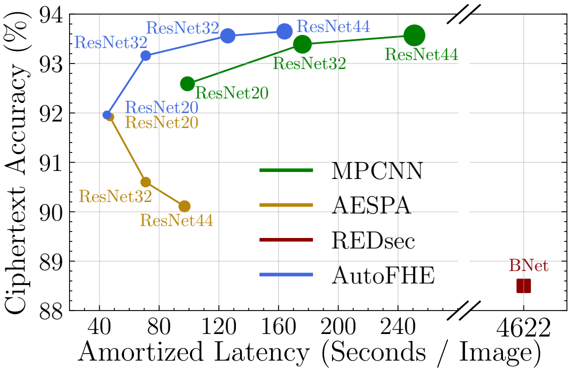

Figure 5: Comparison of trade-offs between ciphertext accuracy and amortized latency on encrypted CIFAR10. The latency is evaluated on Amazon AWS r5.24large using 96 threads. RNS-CKKS approaches take 96 encrypted images as input, while REDsec processes one image at a time. Circle size is proportional to the number of bootstrapping operations.

Minimax composite polynomial approximation of ReLU and reports high ciphertext accuracy (refer to Appendix A). AESPA applies degree 2 Hermite polynomial to replace ReLU, which reduces the multiplicative depth of polynomials and greatly reduces the consumption of bootstrapping. Because AESPA was originally proposed under secure MPC, we implement AESPA under RNS-CKKS (refer to Appendix B). We use the same training parameters for MPCNN, AESPA and AutoFHE. We estimate scaling parameters of MPCNN on training datasets: 21.26 (ResNet20), 21.99 (ResNet32), 17.80 (ResNet44), 29.82 (VGG11) on CIFAR10 dataset and 63.40 (ResNet32) and 54.97 (VGG11) on CIFAR100.

TFHE BASELINE: The fast fully homomorphic encryption scheme over the torus (TFHE) [11] provides very fast bootstrapping by using bootstrapped binary gates. REDsec applies efficient ternary networks (TNNs) under TFHE. REDsec reports performance of BNet$_S$ and BNet on CIFAR10 under CPU TFHE [54]. BNet$_S$ and BNet observe ciphertext accuracy 81.9% and 88.5%, and latency 1,081 and 4,622 seconds per image on CIFAR10 dataset on AWS r5.24large instance using 96 CPU threads [18]. The TFHE cryptographic parameters used by REDsec satisfy 128 bits of security [18]. Please note that REDsec uses a different parallel acceleration strategy. REDsec takes advantage of all 96 CPU threads to process one encrypted image, while RNS-CKKS approaches allocate one thread to each image. Therefore, REDsec has the same latency and amortized latency. When we evaluate the latency of AutoFHE, MPCNN and AESPA on AWS r5.24large, we input 96 encrypted images using 96 CPU threads. The amortized latency is $\frac{1}{96}\times$ as the latency under RNS-CKKS. So, the amortized latency of REDsec and RNS-CKKS approaches is comparable.

4.1 AutoFHE under RNS-CKKS

Trade-offs under FHE: Figure 5 shows trade-offs between ciphertext accuracy and amortized latency on CIFAR10 dataset. We benchmark the performance of ResNet and VGGNet backbones on CIFAR10 and CIFAR100 datasets, as shown in Table 4. For TFHE baseline REDsec, it reported 81.9% ciphertext accuracy with latency 1,081s per image and 88.5% ciphertext accuracy with latency 4,622s per image on CIFAR10 [18]. We have the following observations:

> RNS-CKKS *vs* TFHE
>
> Neural networks under RNS-CKKS significantly outperform those under TFHE in terms of both ciphertext accuracy and latency.

From Figure 5, REDsec shows lower ciphertext accuracy because ternary neural networks (TNNs) enormously compress models [38] compared to real-valued models used by RNS-CKKS approaches. Real-valued networks have much better representation learning ability than TNNs. In secure inference of neural networks, REDsec has to evaluate millions to billions of gates [18]. Although evaluating one gate is very fast (10 $\sim$ 13 ms) [18] on TFHE [54], the latency of the whole network is still extremely high. The most efficient solution (ResNet20 with 5 bootstrapping operations) of **AutoFHE** reports **91.96**% ciphertext accuracy and **45s** latency. Compared to REDsec BNet$_S$ and BNet, AutoFHE improves ciphertext accuracy by **+10.06**% and **+3.46**%, respectively, while being **24**× and **103**× faster in terms of amortized latency.

> High-Degree *vs* Low-Degree *vs* Mixed-Degree
>
> High-degree polynomials are suitable for shallow and deep neural networks but suffer from high latency. Low-degree polynomials can effectively accelerate ciphertext inference but at the cost of a significant drop in accuracy. Mixed-degree layerwise polynomials (AutoFHE) lead to a significantly better trade-off between accuracy and latency.

From Figure 5 and Table 4, we observe that high-degree MPCNN is able to preserve plaintext accuracy although it consumes more levels (14) for AppReLU. Since MPCNN focuses on approximating the ReLU function, it can use weights from ReLU networks and does not suffer from exploding activations and gradients in the forward and backward passes of data through the network. In summary, MPCNN is a plug-in approach with high accuracy and high latency. Low-degree AESPA reduces level consumption of each polynomial by 12 compared to MPCNN. AESPA polynomials only consume 26% and 22% of the bootstrapping operations in MPCNN for ResNet and VGG, respectively. However, AESPA achieves this at the cost of large drops in accuracy, 0.7% $\sim$ 3.6% on

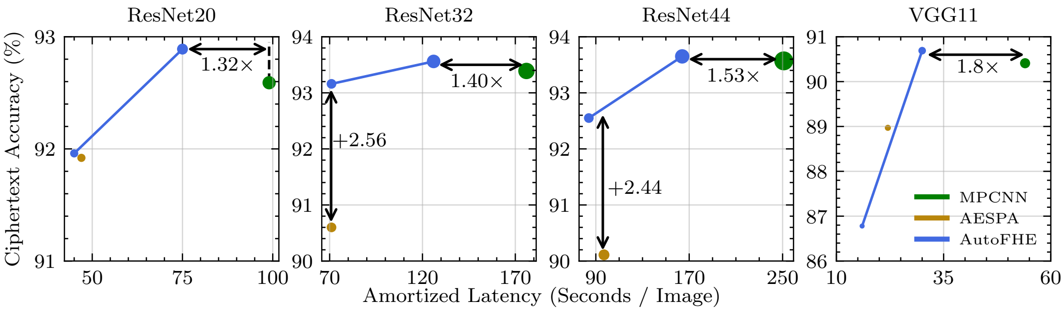

Figure 6: Trade-offs between ciphertext accuracy and amortized latency of ResNet and VGG backbones on CIFAR10.

Dataset	Backbone			MPCNN [33]				AESPA [46]				**AutoFHE**			
	Network	Params	Plain Acc%	Bootstrapping	Cipher Acc%	Latency	Amortized Latency	Bootstrapping	Cipher Acc%	Latency	Amortized Latency	Bootstrapping	Cipher Acc%	Latency	Amortized Latency
CIFAR10	ResNet20	269K	92.66	18	92.59	9,481s	99s	5	91.92	4,554s	47s	5 11	91.96 92.89	4,295s 7,191s	45s 75s
	ResNet32	464K	93.46	30	93.39	16,910s	176s	8	90.60	6,841s	71s	8 19	93.16 93.56	6,860s 12,111s	71s 126s
	ResNet44	658K	93.72	42	93.57	24,082s	251s	11	90.11	9,345s	97s	8 22	92.55 93.65	8,078s 15,769s	84s 164s
	VGG11	123K	90.53	9	90.41	5,221s	54s	2	88.97	2,112s	22s	1 4	86.78 90.69	1,515s 2,879s	16s 30s
CIFAR100	ResNet32	472K	70.81	30	70.59	15,693s	163s	8	68.43	7,171s	75s	16	71.34	10,969s	114s
	VGG11	129K	64.64	9	63.95	5,148s	54s	2	62.90	2,225s	23s	7	64.31	4,562s	48s

Table 4: AutoFHE under the RNS-CKKS scheme. Latency for 96 images is evaluated on AWS r5.24large using 96 threads. Amortized latency is the average latency of each image. We report the ciphertext accuracy of 10,000 encrypted validation images. For CIFAR10, we select two solutions with different bootstrapping for each backbone network. Color key: highest ciphertext accuracy; lowest (amortized) latency; smallest number of bootstrapping operations.

Dataset	Backbone	MPCNN [33]				AESPA [46]				**AutoFHE**			
		#Boot	Linear(s)	AppReLU(s)	Boot(s)	#Boot	Linear(s)	HerPN(s)	Boot(s)	#Boot	Linear(s)	EvoReLU(s)	Boot(s)
CIFAR10	ResNet20	18	1180 ± 32	1067 ± 45	7138 ± 95	5	2344 ± 9	54 ± 1	2108 ± 14	5 11	2014 ± 11 2142 ± 52	115 ± 3 388 ± 18	2092 ± 37 4473 ± 71
	ResNet32	30	2104 ± 84	1708 ± 103	12304 ± 506	8	3582 ± 173	79 ± 2	3234 ± 76	8 19	3540 ± 41 2962 ± 95	38 ± 1 646 ± 20	3107 ± 57 8185 ± 142
	ResNet44	42	3084 ± 93	2385 ± 59	18071 ± 433	11	4683 ± 19	110 ± 2	4393 ± 105	8 22	4595 ± 21 4447 ± 152	42 ± 1 806 ± 41	3265 ± 94 10256 ± 184
	VGG11	9	676 ± 29	639 ± 24	3809 ± 92	2	1281 ± 50	28 ± 1	835 ± 8	1 4	1056 ± 6 1019 ± 7	14 ± 1 172 ± 3	402 ± 21 1661 ± 4
CIFAR100	ResNet32	30	2062 ± 64	1621 ± 51	11815 ± 123	8	3754 ± 20	79 ± 2	3225 ± 78	16	3602 ± 25	620 ± 16	6528 ± 131
	VGG11	9	685 ± 17	645 ± 24	3752 ± 82	2	1330 ± 6	28 ± 1	835 ± 9	7	1145 ± 15	338 ± 18	3027 ± 39

Table 5: Latency of operations under the RNS-CKKS scheme. We report the mean and standard deviation of latency of 96 images in Table 4. Linear layers include Conv, BN, Downsample, AvgPool, and FC.

CIFAR10 and 1.7% ∼ 2.4% on CIFAR100 compared to the corresponding plaintext backbone models.

AutoFHE takes advantage of layerwise mixed-degree polynomials to reduce bootstrapping consumption while maintaining high accuracy. From Figure 6 and Table 4, AutoFHE shows a better trade-off between ciphertext accuracy and latency than MPCNN and AESPA. Compared to MPCNN on CIFAR10, **AutoFHE** accelerates encrypted image inference by **1.32× ∼ 1.8×** while improving accuracy by **+0.08% ∼ 0.3%** in comparison to MPCNN. Similarly, on CIFAR100, **AutoFHE** speeds up inference by **1.1× ∼ 1.4×** while increasing accuracy by **+0.36% ∼ 0.75%**. From Figure 6, compared to AESPA, AutoFHE improves ciphertext accuracy by **+2.56%** and **+2.44%** for ResNet32 and ResNet44 backbones on CIFAR10 with similar amortized latency. However, AESPA's low-degree polynomials lead to an increasing drop in accuracy with model depth. This starkly contrasts with plaintext models, where deeper models are known to improve performance. AutoFHE can improve performance with depth while maintaining the same latency as AESPA. As such, we observe from Figure 5 and Table 4 that AutoFHE enjoys a much better trade-off between accuracy and latency.

In summary, the challenge of navigating the vast joint design space of polynomial approximations of non-linear activation functions and homomorphic evaluation architectures limits manual approaches like MPCNN and AESPA to simplify solutions like approximating non-linear activation functions and uniform placement of bootstrapping operations. In contrast, AutoFHE algorithmically navigates the design space and identifies solutions that significantly *pareto dominate* manual approaches in accuracy, latency, or both.

Approximation *vs* Training *vs* Adaptation

High-precision function approximation can preserve plaintext accuracy without training, while low-precision function approximation with training leads to a loss in accuracy. AutoFHE is a hybrid method that inherits the representation learning ability of ReLU networks and adapts the network's learnable weights to layerwise polynomials.

On the one hand, MPCNN has a drop in accuracy of $0.07 \sim 0.15\%$ and $0.22 \sim 0.69\%$ compared to plaintext backbone models on CIFAR10 and CIFAR100, respectively. This demonstrates that high-degree polynomials still introduce slight approximation errors. On the other hand, AESPA has a significant drop in accuracy, especially for deeper networks, which was also observed by AESPA's authors [46]. The results demonstrate that the representation learning ability of (low-degree) polynomial neural networks is inferior to ReLU networks [37]. Unlike AESPA, AutoFHE inherits the representation learning ability from ReLU networks by using pretrained weights and transferring knowledge. Furthermore, we fine-tune polynomial networks using very small learning rates to adapt learnable network weights to layerwise mixed-degree polynomial EvoReLU. Therefore, AutoFHE can achieve the *high-accuracy* of ReLU networks and the *low-latency* of low-degree polynomial networks. As such, compared to MPCNN and AESPA, AutoFHE improves both prediction accuracy and reduces inference latency over all ResNet and VGG backbones.

Operations under RNS-CKKS: Table 5 shows the latency of different operations under RNS-CKKS. AppReLU is a high-degree polynomial, so its evaluation latency is higher than degree 2 HerPN. The latency of EvoReLU is roughly between AppReLU and HerPN. Low-bootstrapping solutions of AutoFHE further speed up evaluation of polynomial compared to HerPN, *e.g.* ResNet32 with eight bootstrapping, ResNet44 with eight bootstrapping and VGG11 with one bootstrapping. In MPCNN, bootstrapping dominates latency with $74 \sim 77\%$ of total inference time. In AESPA, linear layers and bootstrapping operations consume similar runtime.

The latency of AutoFHE is similar to AESPA for low-bootstrapping solutions and to MPCNN for high-bootstrapping solutions. Linear operations of MPCNN are faster than AESPA and AutoFHE since they are being evaluated at a lower level. For example, MPCNN ConvBN always takes level 2 ciphertext as input. The evaluation of low-level ciphertexts is faster than high-level ciphertexts. For AESPA and AutoFHE, polynomial activations (HerPN and EvoReLU) have smaller multiplicative depth, and their linear operations take ciphertexts at higher levels as input. From Table 5, we observe that i) AutoFHE can effectively accelerate neural network inference on RNS-CKKS by removing time-consuming bootstrapping operations; ii) Layerwise mixed-degree EvoReLU effectively explores how to reduce the multiplicative depth of polynomials and further decrease consumption of bootstrapping operations.

4.2 Layerwise AutoFHE

Depth Distribution: To analyze the layerwise mixed-degree EvoReLU discovered by AutoFHE, we study (see Figure 7) the distributions of multiplicative depth for different backbones on CIFAR10. In contrast to the uniform allocation used by MPCNN and AESPA, the optimal layerwise allocation of EvoReLU discovered by AutoFHE is highly non-uniform. Such a distribution is challenging to design manually, thus further motivating the need for automated design of layerwise mixed-degree polynomial approximations of activation functions. From the distribution in Figure 7, we identify the following design principles that can guide the design of polynomial neural networks under RNS-CKKS.

Observation 1

Low and high bootstrapping solutions share a similar distribution of multiplicative depth.

In Figure 7, we provide two solutions with low and high bootstrapping operations for each backbone. These two solutions share similar depth distributions. Specifically, high-degree polynomials are preferred in the same layers of low and high bootstrapping solutions.

Observation 2

Depth distribution is linearly scalable.

Consider the depth distributions of VGG11(4) and ResNet20(5), ResNet20(11) and ResNet32(19) in Figure 7.

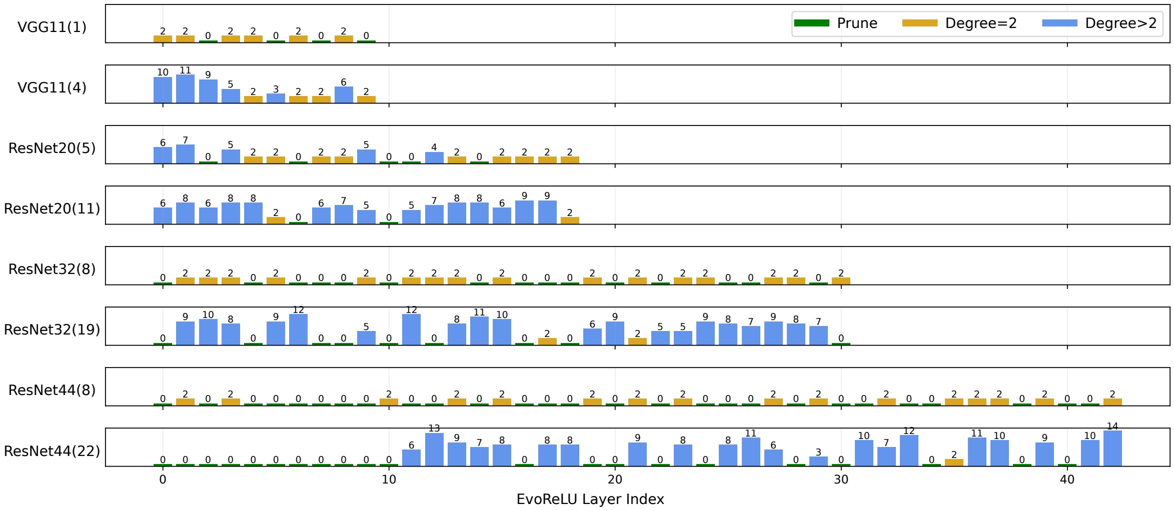

Figure 7: Multiplicative depth of layerwise mixed-degree EvoReLU layers.

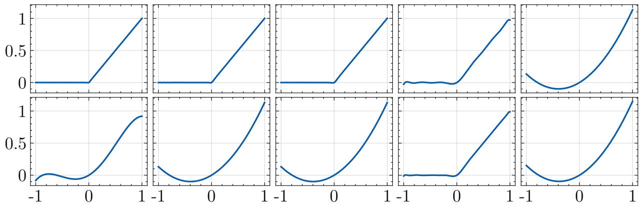

Figure 8: EvoReLU functions of AutoFHE for VGG11 with 4 bootstrapping operations. The top row shows EvoReLU for layer $0 \sim 4$, and the bottom row shows layer $5 \sim 9$.

They have different numbers of layers. If we scale their distributions horizontally to match the number of layers, their depth distributions are very similar. This demonstrates that the number of layers and position of activations are the most important factors affecting the sensitivity of layerwise approximation. Therefore, the depth distributions are linearly scalable to the neural network's depth.

Observation 3

Consecutive linear layers can be integrated into a single operation to reduce multiplicative depth.

Many networks have consecutive linear (depth 0) layers, especially ResNet44(22). In this case, it is possible to integrate successive linear layers into a single linear layer, which decreases the multiplicative depth and removes bootstrapping operations. Table 5 demonstrates that reducing linear operations is an effective way to further accelerate inference.

Layerwise EvoReLU: Figure 8 shows layerwise mixed-degree EvoReLU functions for VGG11 with 4 bootstrapping operations. High-degree EvoReLU functions can precisely approximate ReLU, *e.g.* layer 0, 1 and 2. Medium-degree EvoReLU functions (layer 3, 5, 8) observe oscillation but still are very close to ReLU. Degree 2 EvoReLU is a quadratic function and introduces more approximation errors compared to high-degree and medium-degree functions. For high-degree and medium-degree polynomials, input will be scaled to $[-1,1]$ so we can prevent exploding activations. Figure 8 qualitatively shows that high-degree and medium-degree EvoReLU functions can precisely approximate ReLU, and we can use gradients of ReLU in PAT to prevent exploding gradients during backpropagation. Since degree 2 EvoReLU has a relatively big approximation error, we use SGD rather than approximate gradients from ReLU (refer to Equation 13).

5 Related Work

In this paper, we focus on secure inference under FHE. Secure multiparty computation (MPC) is an alternative approach for secure inference [19, 29, 30, 40, 42, 45, 47]. It is usually employed in a hybrid protocol involving both MPC and HE. MPC primitives include secret sharing, Yao's garbled circuits [2, 59], and Beaver's multiplicative triples [1], *etc*. For example: (1) Gazelle [29] adopts packed additively homomorphic encryption (PAHE) to evaluate linear layers (Conv and FC layers) and garbled circuits to evaluate non-linear layers (ReLU and MaxPooling layers). (2) Delphi [45] uses Garbled circuits to evaluate ReLU, and Beaver's multiplicative triples to evaluate the polynomial approximation x^2 of ReLU. (3) Iron [23] employs the Brakerski-Fan-Vercauteren (BFV) scheme [5, 16] for matrix multiplication. Non-linear operations, like SoftMax, GELU, and LayerNorm, are evaluated through secret sharing.

Secure MPC-based approaches for secure inference must carefully consider the trade-off between computation and communication [29] since both the customer and the Cloud perform computations. In some practical scenarios, sufficient communication and computing resources on the client side may not be available. Pure FHE-based approaches provide customers a *fire-and-forget* [18] service, where they are not involved in the computations and simply wait for the encrypted result. However, FHE-based approaches may have higher latency and memory footprint than secure MPC approaches.

In terms of polynomial neural networks, FHE approaches have to replace *all* ReLU activations with polynomials since FHE only supports multiplications and additions. Secure MPC approaches usually replace only a fraction of ReLUs to reduce online communication and computation costs and retain a few ReLU layers to preserve accuracy, *e.g.* Delphi [45] and SAFENet [42]. These approaches, however, only use low-degree polynomials and observe a significant drop in accuracy when most ReLU activations are replaced. AutoFHE can also be employed for secure MPC schemes by changing the search objective from the number of bootstrapping operations to online communication or computation costs of secure MPC.

6 Conclusion

Non-interactive end-to-end inference of homomorphically encrypted ciphertext images over convolutional networks is an attractive solution for mitigating the security and privacy concerns of cloud-based MLaaS offerings. Adapting CNNs for inference over FHE ciphertexts presents several challenges, including the optimal design of polynomial approximations of non-linear activation functions and associated homomorphic evaluation architecture. Existing solutions primarily rely on manual designs, which are neither scalable nor flexible enough to be applied to any architecture and cater to the needs of different MLaaS customers.

To overcome these challenges, this paper introduced AutoFHE, an automated approach for adapting any convolutional neural network for secure inference under RNS-CKKS. It is a multi-objective search algorithm that generates a set of polynomial networks and their associated homomorphic evaluation architecture under FHE by trading off accuracy and latency. It exploits layerwise mixed-degree polynomial activations across different layers in a network and jointly searches for placement of bootstrapping operations for evaluation under RNS-CKKS. We designed a custom search space for layerwise mixed-degree polynomials and adopted multiple objectives for optimization. We also proposed a combination of search and training algorithms, including multi-objective search algorithm *MOS*, composite polynomial coefficient optimization method *R-CCDE*, and polynomial aware network training strategy *PAT*. We extensively evaluate AutoFHE on ResNets and VGGNets over encrypted CIFAR datasets. Compared to high-degree MPCNN, AutoFHE accelerates inference by $1.32\times \sim 1.8\times$. Compared to low-degree AESPA, AutoFHE improves accuracy by up to 2.56%. Finally, models under RNS-CKKS (AutoFHE) accelerate inference by $103\times$ and improve accuracy by 3.46% compared to models under TFHE (REDsec).

Our results demonstrate the effectiveness of automated search-based algorithms in navigating the large search space of adapting convolution neural networks for inference over FHE ciphertexts and discovering networks that Pareto-dominate manually designed ones. In summary, an integrated and automated design of polynomial approximations and homomorphic evaluation architecture is an effective and flexible approach for seamlessly adapting CNNs for inference on FHE ciphertexts.

References

[1] Donald Beaver. Precomputing oblivious transfer. In *Annual International Cryptology Conference*, pages 97–109, 1995. https://link.springer.com/chapter/10.1007/3-540-44750-4_8.

[2] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 784–796, 2012. https://eprint.iacr.org/2012/265.pdf.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. https://arxiv.org/pdf/1308.3432.pdf.

[4] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 587–617, 2021. https://link.springer.com/chapter/10.1007/978-3-030-77870-5_21.

[5] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886, 2012. https://link.springer.com/chapter/10.1007/978-3-642-32009-5_50.

[6] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821, 2019. http://proceedings.mlr.press/v97/brutzkus19a/brutzkus19a.pdf.

[7] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384,

2018. https://link.springer.com/chapter/10.1007/978-3-319-78381-9_14.

[8] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368, 2018. https://link.springer.com/chapter/10.1007/978-3-030-10970-7_16.

[9] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret LWE. *IEEE Access*, 7:89497–89506, 2019. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8747481.

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437, 2017. https://link.springer.com/chapter/10.1007/978-3-319-70694-8_15.

[11] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. https://eprint.iacr.org/2018/421.pdf.

[12] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster CryptoNets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018. https://arxiv.org/pdf/1811.09953.pdf.

[13] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=996017.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. https://arxiv.org/abs/1810.04805.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. https://openreview.net/pdf?id=YicbFdNTTy.

[16] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012. https://ia.cr/2012/144.

[17] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. https://www.nature.com/articles/s41586-022%20-05172-4.

[18] Lars Wolfgang Folkerts, Charles Gouert, and Nektarios Georgios Tsoutsos. REDsec: Running encrypted discretized neural networks in seconds. In *Network and Distributed System Security Symposium*, 2023. https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_f34_paper.pdf.

[19] Zahra Ghodsi, Nandan Kumar Jha, Brandon Reagen, and Siddharth Garg. Circa: Stochastic relus for private deep learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 2241–2252, 2021. https://proceedings.neurips.cc/paper/2021/file/11eba2991cc62daa4a85be5c0cfdae97-Paper.pdf.

[20] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016. http://proceedings.mlr.press/v48/gilad-bachrach16.pdf.

[21] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, volume 1, pages 69–93. Elsevier, 1991. https://www.sciencedirect.com/science/article/abs/pii/B9780080506845500082.

[22] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. https://arxiv.org/abs/1706.02677.

[23] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems*, volume 35, pages 15718–15731, 2022. https://openreview.net/forum?id=deyqjpcTfsG.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf.

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. https://arxiv.org/abs/1503.02531.

[26] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018. https://openaccess.thecvf.com/content_cvpr_2018/papers/Jacob_Quantization_and_Training_CVPR_2018_paper.pdf.

[27] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. DeepReDuce: ReLU reduction for fast private inference. In *International Conference on Machine Learning*, pages 4839–4849, 2021. http://proceedings.mlr.press/v139/jha21a.html.

[28] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. https://www.nature.com/articles/s41586-021-03819-2.

[29] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium*, pages 1651–1669, 2018. https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-juvekar.pdf.

[30] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure multi-party computation meets machine learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973, 2021. https://proceedings.neurips.cc/paper/2021/file/2754518221cfbc8d25c13a06a4cb8421-Paper.pdf.

[31] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[32] Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9517029.

[33] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422, 2022. https://proceedings.mlr.press/v162/lee22e/lee22e.pdf.

[34] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9734024.

[35] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 618–647, 2021. https://eprint.iacr.org/2020/552.pdf.

[36] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. Precise approximation of convolutional neural networks for homomorphically encrypted data. *arXiv preprint arXiv:2105.10879*, 2021. https://arxiv.org/pdf/2105.10879.pdf.

[37] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. https://www.sciencedirect.com/science/article/pii/S0893608005801315.

[38] Fengfu Li, Bin Liu, Xiaoxing Wang, Bo Zhang, and Junchi Yan. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016. https://arxiv.org/pdf/1605.04711.pdf.

[39] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018. https://openreview.net/pdf?id=S1eYHoC5FX.

[40] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017. https://eprint.iacr.org/2017/452.pdf.

[41] Qian Lou and Lei Jiang. HEMET: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International Conference on Machine Learning*, pages 7102–7110, 2021. http://proceedings.mlr.press/v139/lou21a/lou21a.pdf.

[42] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. SAFENet: A secure, accurate and fast neural network inference. In *International Conference on Learning Representations*, 2020. https://openreview.net/pdf?id=Cz3dbFm5u-.

[43] Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, 2018. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8454482.

[44] Yi Mei, Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *ACM Transactions on Mathematical Software*, 42(2):1–24, 2016. https://dl.acm.org/doi/pdf/10.1145/2791291.

[45] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *USENIX Security Symposium*, pages 2505–2522, 2020. https://www.usenix.org/system/files/sec20-mishra_0.pdf.

[46] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. AESPA: Accuracy preserving low-degree polynomial activation for fast private inference. *arXiv preprint arXiv:2201.06699*, 2022. https://arxiv.org/abs/2201.06699.

[47] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SiRnn: a math library for secure rnn inference. In *IEEE Symposium on Security and Privacy*, pages 1003–1020, 2021. https://eprint.iacr.org/2021/459.pdf.

[48] Hafiz Tayyab Rauf, Waqas Haider Khan Bangyal, and M Ikramullah Lali. An adaptive hybrid differential evolution algorithm for continuous optimization and classification problems. *Neural Computing and Applications*, 33(17):10841–10867, 2021. https://link.springer.com/article/10.1007/s00521-021-06216-y.

[49] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. https://www.nature.com/articles/s41586-020-03051-4.

[50] SEAL. Microsoft SEAL (3.6). *Microsoft Research*, 2020. https://github.com/Microsoft/SEAL.

[51] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. https://www.science.org/doi/full/10.1126/science.aar6404.

[52] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994. https://ieeexplore.ieee.org/abstract/document/6791727.

[53] Mingxing Tan and Quoc Le. EfficientNetV2: Smaller models and faster training. In *International Conference on Machine Learning*, volume 139, pages 10096–10106, 2021.

[54] TFHE. Tfhe v1.1. 2020. https://github.com/tfhe/tfhe.

[55] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. MLP-Mixer: An all-mlp architecture for vision. In *Advances in Neural Information Processing Systems*, volume 34, pages 24261–24272, 2021. https://proceedings.neurips.cc/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[57] Lilian Weng. How to train really large models on many gpus? *lilianweng.github.io*, 2021. https://lilianweng.github.io/posts/2021-09-25-train-large/.

[58] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008. https://www.sciencedirect.com/science/article/pii/S002002550800073X.

[59] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4568207.

[60] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. https://arxiv.org/abs/1611.01578.

A MPCNN under RNS-CKKS

MPCNN [33] is the state-of-the-art framework for homomorphically evaluating deep CNNs on encrypted data under RNS-CKKS with high accuracy. Its salient features are as follows.

(1) COMPACT PACKING: All channels of a tensor are packed into a single ciphertext via multiplexed packing. Furthermore, multiplexed parallel (MP) convolution was proposed to process the ciphertext efficiently. Multiplexed packing can effectively avoid wasting slots due to strided convolutions.

(2) HOMOMORPHIC EVALUATION ARCHITECTURE: To refresh zero-level ciphertexts, bootstrapping operations are placed after every ConvBN (as shown in Figure 2), except for the first one. This hand-crafted homomorphic evaluation architecture for ResNets is determined by the choice of cryptographic parameters, the level consumption of operations, and ResNet's architectures.

(3) APPRELU: It replaces all ReLUs with the same high-degree Minimax composite polynomial [32, 36] of degrees $\{15,15,27\}$. By noting that $\mathrm{ReLU}(x) = x\cdot(0.5+0.5\cdot\mathrm{sgn}(x))$, where $\mathrm{sgn}(x)$ is the sign function, the approximated ReLU (AppReLU) is modeled as $\mathrm{AppReLU}(x) = x\cdot(0.5+0.5\cdot p_\alpha(x)), x\in[-1,1]$. $p_\alpha(x)$ is the composite Minimax polynomial. The precision α is defined as $|p_\alpha(x)-\mathrm{sgn(x)}|\leq 2^{-\alpha}$. AppReLU is expanded to arbitrary domains $x\in[-B,B]$ of pre-activations in CNNs by scaling it as $B\cdot\mathrm{AppReLU}(x/B)$. B can be estimated on the training dataset.

(4) CRYPTOGRAPHIC PARAMETERS: MPCNN sets $N=2^{16}$, $L=30$ and Hamming weight $h=192$. Base modulus, special modulus, and bootstrapping modulus are set to 51 bits, while default modulus is set to 46 bits. These cryptographic parameters satisfy 128 bits of security [9].

(5) MULTIPLICATIVE DEPTH: The multiplicative depth of Bootstrapping (i.e., K), AppReLU, ConvBN, DownSampling, AvgPool, FC layers are 14, 14, 2, 1, 1, 1 respectively. Statistically, when using MPCNN to homomorphically evaluate ResNet-18/32/44 on CIFAR10 or CIFAR100, AppReLUs consume $\sim 50\%$ of total levels, and bootstrapping operations consume $> 70\%$ of latency.

B AESPA under RNS-CKKS

AESPA [46] proposes HerPN to replace ReLU and Batchnorm. HerPN is the approximation of ReLU using Hermite polynomials followed by basis-wise normalization. The Hermite approximation is $f(x) = \sum_{i=0}^{\infty}\hat{f}_i h_i(x)$ where normalized probabilist's Hermite polynomial $h_i(x) = H_n(x)/\sqrt{n!}$ and coefficient $\hat{f}_i = \int_{-\infty}^{\infty} f(x)\overline{h_i(x)}e^{-x^2/2}$. Given tensor $\boldsymbol{x}\in\mathbb{R}^{N\times C\times H\times W}$, HerPN is defined as:

$$f(\boldsymbol{x}) = \boldsymbol{\gamma}\sum_{i=0}^{d}\hat{f}_i\frac{h_i(\boldsymbol{x})-\boldsymbol{\mu}_i}{\sqrt{\boldsymbol{\sigma}_i^2+\varepsilon}}+\boldsymbol{\beta} \tag{14}$$

where mean $\boldsymbol{\mu}_i\in\mathbb{R}^C$ and standard variance $\boldsymbol{\sigma}_i\in\mathbb{R}^C$ are computed over output of each Hermite polynomial, $h_i(\boldsymbol{x})$. $\boldsymbol{\gamma}\in\mathbb{R}^{\mathbb{C}}$ and $\boldsymbol{\beta}\in\mathbb{R}^C$ are learnable parameters. AESPA uses the first three Hermite bases, *i.e.*, $d=2$. Hermite bases and coefficients are:

$$h_0(x)=1 \qquad h_1(x)=x \qquad h_2(x)=\frac{x^2-1}{\sqrt{2}} \tag{15}$$

$$\hat{f}_0=\frac{1}{\sqrt{2\pi}} \qquad \hat{f}_1=\frac{1}{2} \qquad \hat{f}_3=\frac{1}{\sqrt{4\pi}} \tag{16}$$

HerPN can be cast as a second-degree polynomial to reduce multiplicative depth, namely:

$$\begin{aligned} f(\boldsymbol{x}) = {} & \frac{\boldsymbol{\gamma}}{\sqrt{8\pi(\boldsymbol{\sigma}_2^2+\varepsilon)}}\boldsymbol{x}^2 + \frac{\boldsymbol{\gamma}}{2\sqrt{\boldsymbol{\sigma}_1^2+\varepsilon}}\boldsymbol{x}+\boldsymbol{\beta}+ \\ & \boldsymbol{\gamma}\left[\frac{1-\boldsymbol{\mu}_0}{\sqrt{2\pi(\boldsymbol{\sigma}_0^2+\varepsilon)}} - \frac{\boldsymbol{\mu}_1}{2\sqrt{\boldsymbol{\sigma}_1^2+\varepsilon}} - \frac{1+\sqrt{2}\boldsymbol{\mu}_2}{\sqrt{8\pi(\boldsymbol{\sigma}_2^2+\varepsilon)}}\right] \end{aligned} \tag{17}$$

From Equation 17, the depth of HerPN is 2. Hence, 4 Conv-HerPN layers should be followed by one bootstrapping operation. AESPA was originally proposed for secure MPC [46]. Our paper adopts AESPA as a low-degree baseline under RNS-CKKS. We design the homomorphic evaluation architecture of AESPA under RNS-CKKS as shown in Figure 2. We build its C++ implementation on top of MPCNN by using its implementations of Conv, BN, Downsample, AvgPool, and FC layers.