AutoFHE: Automated Adaption of CNNs for Efficient Evaluation over FHE

Wei Ao AOWEI@MSU.EDU
Computer Science and Engineering
Michigan State University
East Lansing, MI 48823, USA

Vishnu Boddeti VISHNU@MSU.EDU
Computer Science and Engineering
Michigan State University
East Lansing, MI 48823, USA

Abstract

Secure inference of deep convolutional neural networks (CNNs) was recently demonstrated under RNS-CKKS. The state-of-the-art solution uses a high-order composite polynomial to approximate all ReLUs. However, it results in prohibitively high latency because bootstrapping is required to refresh zero-level ciphertext after every Conv-BN layer. To accelerate inference of CNNs over FHE and automatically design homomorphic evaluation architectures of CNNs, we propose AutoFHE: a bi-level multi-objective optimization framework to automatically adapt standard CNNs to polynomial CNNs. AutoFHE can maximize validation accuracy and minimize the number of bootstrapping operations by assigning layerwise polynomial activations and searching for the placement of bootstrapping operations. As a result, AutoFHE can generate diverse solutions spanning the trade-off front between accuracy and inference time. Experimental results of ResNets on encrypted CIFAR-10 under RNS-CKKS indicate that in comparison to the state-of-the-art solution, AutoFHE can reduce inference time (50 images on 50 threads) by up to 3,297 seconds (43%) while preserving accuracy (92.68%). AutoFHE also improves the accuracy of ResNet-32 by 0.48% while accelerating inference by 382 seconds (7%).

Keywords: Fully Homomorphic Encryption, Deep Learning, Neural Architecture Search, Automated Machine Learning

1. Introduction

MLaaS, machine learning as a service, is an exponentially growing market. Commercial MLaaS platforms, like Amazon AWS, Google GCP, and Microsoft Azure, have proliferated. As a novel Cloud computing service, MLaaS is driven by the great success of deep learning on extensive tasks, like vision (Dosovitskiy et al., 2020; He et al., 2016), language (Devlin et al., 2018; Vaswani et al., 2017), game (Schrittwieser et al., 2020; Silver et al., 2018), science (Fawzi et al., 2022; Jumper et al., 2021), and many more. In the scenario of MLaaS, the Cloud (Alice) holds pre-trained 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 any sensitive information. Deep learning models, including *neural architectures* and *pre-trained weights*, are properties of Alice. Alice spends considerable efforts to design neural architectures, like ResNets (He et al., 2016), ViT (Dosovitskiy et al., 2020),

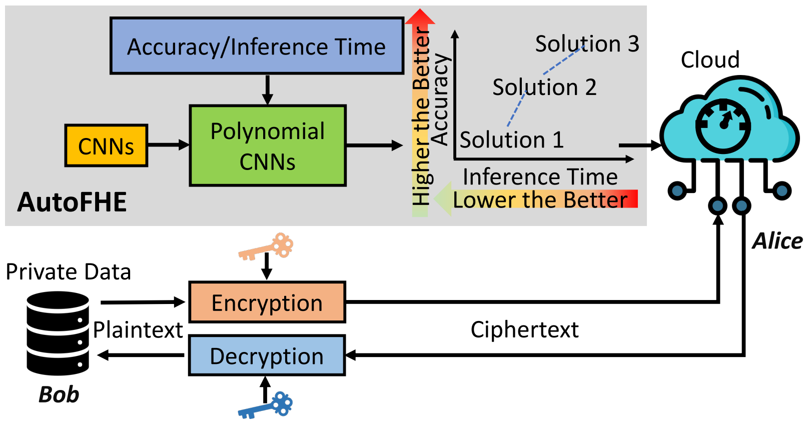

Figure 1: AutoFHE is a bi-level multi-objective optimization framework to adapt standard CNNs into polynomial CNNs automatically. AutoFHE takes as input a neural network with ReLUs and generates a diverse set of polynomial networks by trading off accuracy and inference time. We can deploy AutoFHE solutions on the Cloud to provide different kinds of service to customers.

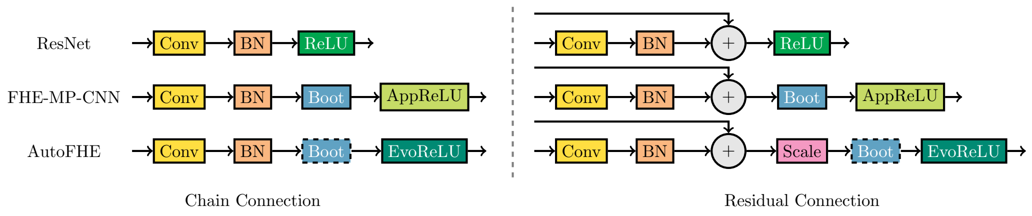

Figure 2: Homomorphic evaluation architectures of the chain and residual connections. Top row: standard ResNet Conv-BN-ReLU triplet (He et al., 2016). Middle row: FHE-MP-CNN. Bottom row: AutoFHE, where dashed rectangles are placement of bootstrapping layers to search.

and MLP-Mixer (Tolstikhin et al., 2021) and consumes huge computational resources to search for novel neural architectures (Liu et al., 2018; Zoph and Le, 2016) or train network weights (Goyal et al., 2017; Weng, 2021).

Fully Homomorphic Encryption: Secure inference of deep learning models under homomorphic encryption (HE)(Brutzkus et al., 2019; Gilad-Bachrach et al., 2016; Lou and Jiang, 2021) or fully homomorphic encryption (FHE) (Lee et al., 2022a,b) is a promising solution for resolving security concerns between Alice and Bob in the context of MLaaS. FHE enables us to evaluate a circuit with arbitrary depth and shows potential to embrace modern deep CNNs (Lee et al., 2022a). 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 and yields an encrypted result. Finally, Bob uses the secret key to decrypt the encrypted result. Under FHE, Bob cannot learn Alice's neural architectures and pre-trained weights, while Alice is also not exposed to both Bob's data and result.

Polynomial CNNs: Non-arithmetic activation functions, such as $\mathrm{ReLU}(x) = \max(x, 0)$, are widely used in modern CNNs to learn non-linear features. For example, Residual Networks (He et al., 2016) are composed of Conv-BN-ReLU triplets (He et al., 2016) as shown in the top row of Figure 2. Since FHE only supports multiplications and additions, ReLUs must be replaced by polynomial approximations to evaluate CNNs under FHE. Existing methods to generate polynomial CNNs can be categorized as **training**-based methods at the network level and **approximation**-based methods at the function level.

Training-based methods adopt *low-degree* (typically ≤ 3) polynomials (Brutzkus et al., 2019; Chou et al., 2018; Gilad-Bachrach et al., 2016; Lou and Jiang, 2021; Lou et al., 2020; Mishra et al., 2020; Park et al., 2022) to substitute non-arithmetic activation functions and then *train* polynomial neural networks *from scratch*. CryptoNets (Gilad-Bachrach et al., 2016), LoLa (Brutzkus et al., 2019) and Delphi (Mishra et al., 2020) employ a simple square activation function x^2. Faster CryptoNets (Chou et al., 2018) exploit more accurate low-degree approximation $2^{-3}x^2 + 2^{-1}x + 2^{-2}$. SAFENet (Lou et al., 2020) adopts $a_1x^3 + a_2x^2 + a_3x + a_4$ or $b_1x^2 + b_2x + b_3$ and HEMET (Lou and Jiang, 2021) 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). Although Delphi (Mishra et al., 2020) and SAFENet are based on secure MPC, they use polynomials to substitute a fraction of the ReLUs to decrease online communication and computation costs. However, polynomial layers often lead to unstable training since they may dramatically amplify gradients during back-propagation. Such gradient explosion was observed in prior works (Lou et al., 2020; Mishra et al., 2020).

As such, training-based methods suffer from a *dilemma*. On the one hand, since low-degree polynomials cannot precisely approximate ReLUs, we have to train polynomial networks 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, training-based methods report lower accuracy than ReLU-based networks, *e.g.* on CIFAR-10 HEMET (Lou and Jiang, 2021) and SAFENet (Lou et al., 2020) report 83.7% and 88.9% Top-1 accuracy, respectively. Training-based methods can save depth consumption due to low-degree polynomials and evaluate polynomial networks under leveled HE. However, they cannot inherit pre-trained weights from ReLU-based networks and report lower accuracy.

Approximation-based methods use high-degree polynomials to approximate the ReLU function precisely. So, approximation-based methods do not need to train polynomial networks from scratch and can inherit pre-trained weights from ReLU-based networks. One representative polynomial approximation of ReLU is Minimax composite polynomials (Lee et al., 2021a,c). 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 the precision of approximation $|p_\alpha(x) - \mathrm{sgn(x)}| \leq 2^{-\alpha}$. 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 levels and require bootstrapping, leading to a high computational burden.

FHE-MP-CNN (Lee et al., 2022a), the state-of-the-art method for secure inference of CNNs under RNS-CKKS, adopts Minimax composite polynomials with a precision of $\alpha = 13$, leading to prediction accuracy comparable to ResNets, thanks to the precise approximation of Minimax composite polynomials. However, FHE-MP-CNN *is still limited in four respects:*

- The high-precision AppReLU only considers function-level approximation and neglects the potential for end-to-end optimization of the complete network response.
- The same high-degree AppReLU replaces all ReLUs and consumes **47%** levels. Thus, the potential for using mixed-degree approximations across the different neural network layers is not exploited.
- The ciphertext quickly exhausts levels and uses bootstrapping to refresh the zero-level ciphertext. Bootstrapping operations consume **77%** of inference time.
- The homomorphic evaluation architecture (middle row in Figure 2) lacks flexibility due to the careful design of AppReLUs, cryptographic parameters, placement of bootstrapping operations, and network architectures.

AutoFHE Motivation: To address the limitations of existing secure inference systems of CNNs mentioned above, we propose AutoFHE drawing inspiration from the following observations:

Observation 1

Mixed-degree layerwise polynomial networks can accelerate inference while preserving accuracy.

An intuitive solution to decrease computational burden is to assign mixed-degree layerwise polynomials to different ReLUs across a network by assuming sensitivity to approximation error is varying across different layers. SAFENet (Lou et al., 2020) demonstrates that mixed-degree polynomials can take advantage of layerwise sensitivity. However, current mixed-degree search frameworks are limited in two aspects. **1) 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. **2) Scalarization of multiple objectives:** a weighted sum is used to balance multiple objectives (Lou et al., 2020; Mishra et al., 2020). However, such an approach requires a pre-defined preference to weigh the different objectives. As such, they cannot generate diverse solutions to meet different requirements in a single run.

Observation 2

Approximation-based methods can precisely approximate ReLU, while training-based methods can jointly learn the polynomial coefficients and network weights through gradient descent.

FHE-MP-CNN demonstrates the state-of-the-art accuracy of approximation-based methods. Training-based methods show that lower-degree polynomial networks are relatively trainable. However, **fine-tuning polynomial CNNs with high-degree polynomial approximations of ReLU cannot match the accuracy** of ReLU-based networks (Lee et al., 2021c).

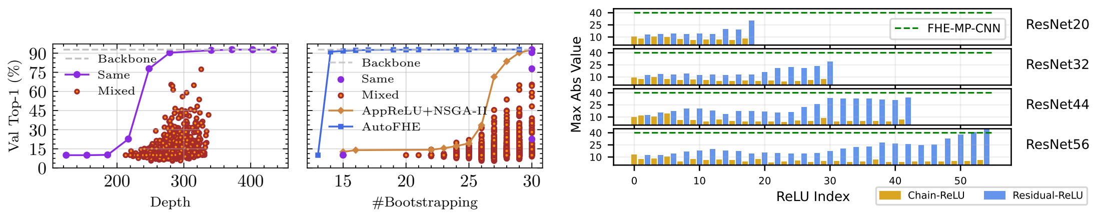

Figure 3: Motivating AutoFHE. Left: depth consumption of AppReLUs based on ResNet-20 backbone on CIFAR-10. The purple line is when the same precision AppReLU is used in all layers, while the red circles show 5000 randomly-sampled combinations of mixed-precision layerwise AppReLUs. Middle: the number of bootstrapping operations where we show trade-offs of the same AppReLU and mixed AppReLUs as in the top left panel. We also show a multi-objective search result using mixed-precision layerwise AppReLUs and the trade-off front of the proposed AutoFHE. Right: distributions of pre-activations (the maximum absolute values) of ResNets on CIFAR-10 where the green line corresponds to B, the scale value of AppReLU in FHE-MP-CNN.

We propose AutoFHE (Figure 1) to **Auto**matically generate polynomial CNNs that span a trade-off front of accuracy and inference latency under **FHE**. It is designed to address the following questions and resolve the associated challenges:

- **Question 1: Can we *scale up* the search space of layerwise polynomials to incorporate both training- and approximation-based methods?** Intuitively, we can find better solutions in a large search space that includes both low-degree training-based and high-degree approximation-based solutions. However, it brings **Challenge 1: How can we *efficiently* search for solutions in the huge search space?**

- **Question 2: Can we *fine-tune* mixed-degree polynomial networks to take advantage of both function approximation and network learning ability?** We want to fine-tune such mixed-degree polynomial networks to improve accuracy after discovering solutions in the search space. However, it brings **Challenge 2: How can we *effectively* prevent gradients from exploding when fine-tuning polynomial networks, including low- and high-degree polynomials?**

- **Question 3: Can we *trade off* accuracy and inference time to generate solutions toward the Pareto front to meet different customer requirements?** Unlike existing solutions that use a single objective, we design a multi-objective optimization framework to generate solutions toward the Pareto front to satisfy different requirements. The bottleneck of evaluating CNNs under FHE is bootstrapping. The number of bootstrapping operations is a countable objective and can be used during the search. However, it brings **Challenge 3: How can we *automatically* design homomorphic evaluation architectures for every possible solution?**

Case Study: We consider two plausible solutions to trade off accuracy and computational burden of FHE-MP-CNN (Figure 3). (i) **Same Precision AppReLU:** We replace all ReLU layers with AppReLU of a given precision. We can trade off (purple line in the left

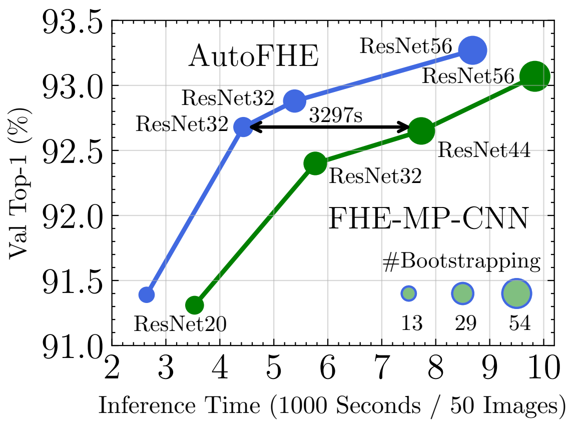

Figure 4: Trade-offs of AutoFHE *versus* FHE-MP-CNN on encrypted CIFAR-10 under the RNS-CKKS FHE scheme.

panel) *accuracy* and *depth* consumption using AppReLU with different precision. However, as the middle panel shows, these solutions (purple dots) do not necessarily translate to a trade-off between *accuracy* and the number of *bootstrapping* operations due to many wasted levels. All the trade-off solutions collapse to either 15 or 30 bootstrapping operations. (ii) **Mixed-Precision AppReLU:** Each ReLU layer in the network can be replaced by AppReLU of *any* precision. We randomly sample 5,000 combinations of mixed-precision layerwise AppReLUs and show (red dots) their depth consumption and the number of bootstrapping operations in the left and middle panels, respectively. Observe that layerwise mixed-precision AppReLU leads to a better trade-off between accuracy and the number of bootstrapping operations. However, FHE-MP-CNN neglects the layerwise sensitivity (range) of ReLU pre-activations (the right panel shows the distribution of the layerwise maximum absolute value of pre-activation) and uses AppReLU, which is optimized for a ReLU with a large pre-activation range. Therefore, the trade-off of mixed-precision layerwise AppReLU optimized by a multi-objective search algorithm NSGA-II (Deb et al., 2002) is still inferior to AutoFHE by a significant margin.

Contributions: In this paper, we relax the design choices of existing secure inference systems of CNNs under FHE and accelerate the inference of CNNs over homomorphically encrypted data while maximizing performance. The central premise behind our approach is to *directly optimize the end-to-end function represented by the network instead of optimizing the function represented by the activation function.* This idea allows us to exploit the varying sensitivity of activation function approximation across different layers in a network. Therefore, theoretically, evolving layerwise polynomial approximations of ReLUs (EvoReLU) should reduce the total multiplicative depth. We jointly search for placement of bootstrapping (bottom row in Figure 2) and thus reduce the number of time-consuming bootstrapping operations so we can finally minimize inference time on encrypted data. From a **system** perspective, our contributions are three-fold:

1. **Search Space:** We design search space to include all possible polynomial networks to enable us to discover better solutions. The search space size is $10^{79} \sim 10^{230}$.

2. **Search Objective:** We formulate the search problem as a bi-level multi-objective optimization. We automatically generate diverse polynomial networks spanning the trade-off front between accuracy and latency.

3. **Search Algorithms:** We propose a combination of search and training algorithms to search over large search spaces efficiently, optimize coefficients of arbitrary polynomials, and fine-tune mixed-degree polynomial networks. Specifically, we propose:

   - **MOCoEv**, a multi-objective coevolutionary search algorithm to optimize high-dimensional variables ($114 \sim 330$) in large search space.
   - **R-CCDE**, a gradient-free search algorithm to optimize coefficients of composite polynomials by only using the difference between solutions.
   - **PAT**, a fine-tuning algorithm that enables network weighs aware polynomial activations and avoids exploding gradients.

Experimental results on encrypted CIFAR-10 and CIFAR-100 under RNS-CKKS show that (Figure 4), compared to FHE-MP-CNN, the state-of-the-art approach, AutoFHE shows better trade-offs between accuracy and inference time. On CIFAR-10, AutoFHE reduces inference time (50 images on 50 threads) by up to 3,297 seconds (43%) while preserving the accuracy (92.68%). Specifically, AutoFHE reduces inference time of ResNet-20, ResNet-32 (21 bootstrapping operations), and ResNet-56 by 25%, 23%, and 12%, respectively, while improving accuracy up to 0.28%. AutoFHE also improves the accuracy of ResNet-32 (29 bootstrapping operations) on CIFAR-10 by 0.48% while accelerating inference by 382 seconds (7%). On CIFAR-100, AutoFHE saves inference time by 972 seconds (17%) while preserving accuracy.

2. Preliminaries

RNS-CKKS: The full residue number system (RNS) variant of Cheon-Kim-Kim-Song (RNS-CKKS) (Cheon et al., 2017, 2018b) is a leveled homomorphic encryption (HE) 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 $0 \leq \ell \leq L$. ℓ is a non-negative integer referred to as *level* 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:

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

Bootstrapping: Leveled HE 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 (Cheon et al., 2018a) 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 $\mathcal{R}_{q_0} \cong \mathcal{R}_{q_0} \times \mathcal{R}_{q_1} \times \cdots \times \mathcal{R}_{q_L}$ (Bossuat et al., 2021). Practically, bootstrapping (Cheon et al., 2018a) homomorphically evaluates modular reduction $[\cdot]_Q$ by first approximating it by a scaled sine function, which is further approximated through polynomials (Cheon et al., 2018a; Lee et al., 2021b). Bootstrapping (Bossuat et al., 2021) has four stages: ModRaise, CoeffToSlot, EvalMod, and SlotToCoeff. These operations involve a lot of homomorphic multiplications and rotations, both of which are costly operations, especially the latter. The refreshed ciphertext has level $\ell = L - K$, where K levels are consumed by bootstrapping (Bossuat et al., 2021) for polynomial approximation of modular reduction.

FHE-MP-CNN (Lee et al., 2022a) is the state-of-the-art framework for homomorphically evaluating deep CNNs on encrypted data under RNS-CKKS with high accuracy. Its salient features include 1) *Compact Packing:* All channels of a tensor are packed into a single ciphertext. Furthermore, multiplexed parallel (MP) convolution was proposed to process the ciphertext efficiently. 2) *Homomorphic Evaluation Architecture:* Bootstrapping operations are placed after every Conv-BN (as shown in Figure 2), except for the first one, to refresh zero-level ciphertexts. This hand-crafted homomorphic evaluation architecture for ResNets is determined by choice of cryptographic parameters, the level consumption of operations, and ResNet's architecture. 3) *AppReLU:* It replaces all ReLUs with the same high-order Minimax composite polynomial (Lee et al., 2021a,c) of degrees $\{15, 15, 27\}$. By noting that $\text{ReLU}(x) = x \cdot (0.5 + 0.5 \cdot \text{sgn}(x))$, where $\text{sgn}(x)$ is the sign function, the approximated ReLU (AppReLU) is modeled as $\text{AppReLU}(x) = x \cdot (0.5 + 0.5 \cdot p_\alpha(x)), x \in [-1, 1]$. $p_\alpha(x)$ is the composite Minimax polynomial. The precision α is defined as $|p_\alpha(x) - \text{sgn(x)}| \leq 2^{-\alpha}$. AppReLU is expanded to arbitrary domains $x \in [-B, B]$ of pre-activations in CNNs by scaling it as $B \cdot \text{AppReLU}(x/B)$. However, this reduces approximation precision to $B \cdot 2^{-\alpha}$. To estimate the maximum dynamic range B (40 for CIFAR-10 and 65 for CIFAR-100) of ReLUs, FHE-MP-CNN evaluates the pre-trained network on the training dataset. FHE-MP-CNN uses the same dynamic range B for all polynomials and neglects the uneven distribution of pre-activations as shown in Figure 3. Explicitly accounting for this uneven distribution allows us to use smaller B' and α' but with the same precision, i.e., $B' \cdot 2^{-\alpha'} = B \cdot 2^{-\alpha}$, for $B' < B$ and $\alpha' < \alpha$. 4) *Cryptographic Parameters:* FHE-MP-CNN sets $N = 2^{16}$, $L = 30$ and Hamming weight $h = 192$. Please refer to (Lee et al., 2022a) for the detailed implementation of FHE-MP-CNN and other parameters. These parameters provide 128-bits of security (Cheon et al., 2019). 5) *Depth Consumption:* To reduce level consumption, FHE-MP-CNN integrates scaling parameter B into Conv-BN. The multiplicative depth consumption of Bootstrapping (i.e., K), AppReLU, Conv, DownSampling, AvgPool, FC, and BN layers are 14, 14, 2, 1, 1, 1, 0, respectively. Statistically, when using FHE-MP-CNN to homomorphically evaluate ResNet-18/32/44/56 on CIFAR-10 or CIFAR-100, AppReLUs consume $\sim 47\%$ of total levels, and bootstrapping operations consume $\sim 77\%$ of inference time.

Threat Model: In this paper, we assume the same threat model as FHE-MP-CNN. As discussed in the MLaaS scenario, a customer generates a public key to encrypt the private data under RNS-CKKS and then sends the ciphertext to an untrusted Cloud service provider.

The Cloud uses neural networks to process the ciphertext without decryption, yielding an encrypted result. The customer finally uses the secret key to decrypt the result. The Cloud cannot learn any sensitive information from the customer's data.

3. AutoFHE

Given a neural network $f(\boldsymbol{\omega}_0)$ with pre-trained weight $\boldsymbol{\omega}_0$ and M ReLU layers, AutoFHE generates polynomial networks on a trade-off front by maximizing validation accuracy and minimizing inference time. *During the search*, every solution has a triplet variable, the degree vector of our polynomial approximations of ReLU, the corresponding coefficient vector, and the network trainable weight $\{\boldsymbol{D}, \boldsymbol{\Lambda}, \boldsymbol{\omega}\}$. $\boldsymbol{\omega}$ is initialized with $\boldsymbol{\omega}_0$ and then fine-tuned on the training dataset for only *a few epochs*. We will assign each solution with the minimization objective $\boldsymbol{o} = \{1 - \text{Acc}, \text{Boot}\}$. $1 - \text{Acc}$ is the validation error, and Boot is the number of bootstrapping operations. A possible polynomial network is denoted as $\{\boldsymbol{D}, \boldsymbol{\Lambda}, \boldsymbol{\omega}\} \Rightarrow \boldsymbol{o}$. AutoFHE is a search-based system and comprises of **3.1 search space**, **3.2 search objective** and **3.3 search algorithms**.

3.1 Search Space

EvoReLU is a polynomial function used to replace non-arithmetic ReLU

$$y = \text{EvoReLU}(x) = x \cdot \Big(0.5 + p^d(x)\Big) \tag{2}$$

where $x \in [-1, 1], y \in [0, 1]$. The composite polynomial $p^d(x)$ is

$$p^d(x) = (p_K^{d_K} \circ \cdots \circ p_k^{d_k} \circ \cdots \circ p_1^{d_1})(x), 1 \leq k \leq K \tag{3}$$

The composite polynomial $p^d(x)$ has K sub-polynomial functions and degree $d = \prod_{k=1}^{K}$. This structure for EvoReLU bears similarity to the Minimax composite polynomial in (Lee et al., 2022a, 2021c). However, the objective for optimizing the coefficients is significantly different in AutoFHE.

Representation: We represent the composite polynomial $p^d(x)$ by its degree vector $\boldsymbol{d} = \{d_i\}_{i=1}^{d_K}, d_i \in \mathbb{Z}^+$, and each sub-polynomial $p_k^{d_k}(x)$ as a linear combination of Chebyshev polynomials[1] of degree d_k, i.e.

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

where $\alpha_i \in \mathbb{R}$ and $\beta_k \in \mathbb{R}$. $\mathrm{T}_i(x)$ is the Chebyshev bases of the first kind, α_i are the coefficients for linear combination, and 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:

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

1. AutoFHE is agnostic to the choice of polynomial bases and can use other ones like Hermite polynomials.

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

Table 1: Search variables and options

Backbone	#ReLUs	Dimension of $\boldsymbol{D}$	Search Space Size
ResNet-20	19	114	10^{79}
ResNet-32	31	186	10^{130}
ResNet-44	43	258	10^{180}
ResNet-56	55	330	10^{230}

Table 2: Search space of AutoFHE for ResNet backbones.

A neural network with M ReLU layers needs M EvoReLU polynomial activations. $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_M\}$ is the degree vector of all EvoReLUs, the corresponding coefficient parameters are $\mathbf{\Lambda} = \{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \cdots, \boldsymbol{\lambda}_M\}$.

Homomorphic Evaluation Architecture: The ResNet architecture comprises two types of connections, a chain, and a residual connection, as shown in Figure 2. To extend the domain of EvoReLU from $[-1, 1]$ to $[-B, B]$ but avoid extra depth consumption for scaling, we scale the plaintext weight and bias of BatchNorm by $1/B$ in advance for chain connections. But for residual connections, we cannot integrate the scale $1/B$ into BatchNorm's weight and bias. In this case, we scale the ciphertext output of the residual connection by $1/B$ at the expense of one level. Finally, we integrate B into coefficients of $p_K^{d_K}(x)$ to re-scale the output of EvoReLU by B. Given the pre-activation $x \in [-B, B]$, the scaled EvoReLU with the degree $\boldsymbol{d}$ is parameterized by $\boldsymbol{\lambda}$:

$$y = \text{EvoReLU}(x, \boldsymbol{\lambda}; \boldsymbol{d}) = x \cdot (0.5 + p^{d}(x)) \tag{6}$$

where $x \in [-B, B], y \in [0, B]$. We estimate B values for layerwise EvoReLUs on the training dataset. From Figure 2, FHE-MP-CNN places bootstrapping after every Conv-BN, while AutoFHE will search for placement of bootstrapping operations by adjusting to different depth consumption of layerwise EvoReLUs.

The Depth Consumption of EvoReLU is $1 + \sum_{k=1}^{K} \lceil \log_2(d_k + 1) \rceil$ when using the Baby-Step Giant-Step (BSGS) algorithm (Bossuat et al., 2021; Lee et al., 2021b) to evaluate $p^d(x)$.

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 the polynomials $\mathbf{\Lambda}$. Table 1 shows the options for each of these variables. 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 (Lee et al., 2021a,c)), 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

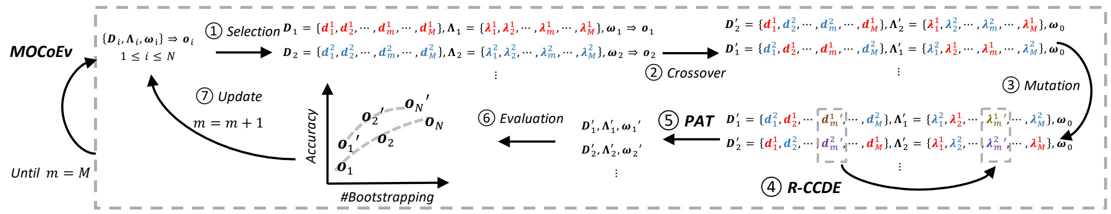

Figure 5: AutoFHE search. The outer search algorithm is MOCoEv. Assume we maintain N solutions during the search. We highlight two solutions in the illustration. We repeat the iteration until $m = M$ to finish one-generation evolution.

models and the corresponding dimension and size of search space for $\boldsymbol{D}$. The extremely large search space ($10^{79} \sim 10^{230}$) makes searching for solutions very challenging.

3.2 Search Objective

Multi-Objective Optimization: Given two solutions with minimization objectives $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$, we want to minimize $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$. If $\boldsymbol{o}_{1,i} \leq \boldsymbol{o}_{2,i}, \forall i \in \{1,2\}$ and $\boldsymbol{o}_{1,j} < \boldsymbol{o}_{2,j}, \exists j \in \{1,2\}$, $\boldsymbol{o}_1$ *dominates* $\boldsymbol{o}_2$ (Deb et al., 2002; Srinivas and Deb, 1994). It means $\boldsymbol{o}_1$ is better than $\boldsymbol{o}_2$. It is denoted as $\boldsymbol{o}_1 \prec \boldsymbol{o}_2$. AutoFHE generates a class of solutions $\boldsymbol{O} = \{\boldsymbol{o}_i\}_{i=1}^{N}$ spanning the trade-off front satisfying $\boldsymbol{o}_i \nprec \boldsymbol{o}_j$ and $\boldsymbol{o}_j \nprec \boldsymbol{o}_i$, $1 \leq i, j \leq N$, $i \neq j$. The optimal $\boldsymbol{O}$ is called Pareto front (Deb et al., 2002; Srinivas and Deb, 1994).
Search Objective: AutoFHE formulates the multi-objective search problem as *a bi-level multi-objective* optimization

$$\begin{aligned} \min_{\boldsymbol{D}} \quad & \{1 - \text{Acc}_{val}\left(f(\boldsymbol{\omega}^*); \boldsymbol{D}, \boldsymbol{\Lambda}(\boldsymbol{D})\right), \text{Boot}(\boldsymbol{D})\} \\ & \boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \mathcal{L}_{train}\left(f(\boldsymbol{\omega}); \boldsymbol{D}, \boldsymbol{\Lambda}(\boldsymbol{D})\right) \end{aligned} \tag{7}$$

where $f(\boldsymbol{\omega})$ is a neural network with M ReLU layers and the trainable network weight $\boldsymbol{\omega}$. The degrees of layerwise EvoReLU's $\boldsymbol{D} = \{\boldsymbol{d}_m\}_{m=1}^{M}$ is the upper-level variable, while the network trainable weight $\boldsymbol{\omega}$ is the lower-level variable. The outer multi-objective minimization formulation $\min_{\boldsymbol{D}} \{1 - \text{Acc}_{val}\left(\boldsymbol{\omega}^*; \boldsymbol{D}, \boldsymbol{\Lambda}(\boldsymbol{D})\right), \text{Boot}(\boldsymbol{D})\}$ for $\boldsymbol{D}$ is to **maximize** the validation accuracy Acc_{val} as well as **minimize** the number of bootstrapping operations. The coefficient vector $\boldsymbol{\Lambda}$ is 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 depth consumption of all EvoReLU's to determine when 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. $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i\}$ has its corresponding network weight $\boldsymbol{\omega}_i$ that can compensate errors introduced by layerwise EvoReLU $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i\}$. We initialize $\boldsymbol{\omega}_i$ with the weight $\boldsymbol{\omega}_0$ from the pre-trained ReLU-based network and then fine-tune the network $f(\boldsymbol{\omega}_i)$ to minimize the training loss $\mathcal{L}_{train}(\boldsymbol{\omega}_i)$ on the training dataset. In summary, the objective in Equation 7 guide the search algorithm to, i) explore layerwise EvoReLU, including its *degrees* and *coefficients*; 2) discover the placement of bootstrapping to work well with EvoReLU; 3) trade off validation accuracy and inference speed to return diverse polynomial networks. To optimize Equation 7, we propose the following algorithms:

- **MOCoEv** is proposed to optimize the outer multi-objective $\min_{\boldsymbol{D}} \{1 - \mathrm{Acc}_{val}\left(f(\boldsymbol{\omega}^*); \boldsymbol{D}, \boldsymbol{\Lambda}(\boldsymbol{D})\right), \mathrm{Boot}(\boldsymbol{D})\}$;
- **R-CCDE** is a gradient-free search algorithm to solve $\boldsymbol{\Lambda}(\boldsymbol{D})$;
- **PAT** is designed to fine-tune polynomial networks with EvoReLUs to minimize $\mathcal{L}_{train}\left(f(\boldsymbol{\omega}); \boldsymbol{D}, \boldsymbol{\Lambda}(\boldsymbol{D})\right)$.

Search Overview: Figure 5 provides a systematic overview of AutoFHE search framework, including MOCoEv, R-CCDE, and PAT. **MOCoEv** is the outer search algorithm to solve the bi-level multi-objective optimization in Equation 7. The current trade-off front is $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \leq i \leq N$. In Figure 5, we highlight two solutions $\{\boldsymbol{D}_1, \boldsymbol{\Lambda}_1, \boldsymbol{\omega}_1\} \Rightarrow \boldsymbol{o}_1$ and $\{\boldsymbol{D}_2, \boldsymbol{\Lambda}_2, \boldsymbol{\omega}_2\} \Rightarrow \boldsymbol{o}_2$. MOCoEV has a *network-level crossover* to combine two solutions to generate two new solutions. We can exchange degrees and coefficients of corresponding EvoReLU layers. However, we cannot inherit network weights $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ because they are fine-tuned for the combinations $\{\boldsymbol{D}_1, \boldsymbol{\Lambda}_1\}$ and $\{\boldsymbol{D}_2, \boldsymbol{\Lambda}_2\}$. So, $\{\boldsymbol{D}_1', \boldsymbol{\Lambda}_1'\}$ and $\{\boldsymbol{D}_2', \boldsymbol{\Lambda}_2'\}$ inherit the network weight $\boldsymbol{\omega}_0$ from the ReLU-based network. Then, the m-layer EvoReLU's are mutated to explore better polynomials locally. **R-CCDE** is a gradient-free *function-level* search algorithm to optimize coefficients. Third, **PAT** is proposed to fine-tune networks with mixed-precision high-degree polynomials. After we obtain $\{\boldsymbol{D}_1', \boldsymbol{\Lambda}_1', \boldsymbol{\omega}_1'\}$ and $\{\boldsymbol{D}_2', \boldsymbol{\Lambda}_2', \boldsymbol{\omega}_2'\}$, we can estimate objectives $\boldsymbol{o}_1'$ and $\boldsymbol{o}_2'$, namely the accuracy and the number of bootstrapping operations. Finally, we get the new trade-off front from $\{\boldsymbol{o}\}_{i=1}^N \cup \{\boldsymbol{o}_j'\}_{j=1}^N$. We repeat the process until we update all EvoReLU layers, namely $m = M$. These M-successive *iterations* are referred to as one *generation*.

3.3 Search Algorithms

3.3.1 MOCoEv

Motivation: Assuming we have N solutions: $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i$, $1 \leq i \leq N$, these solutions span the trade-off front. MOCoEv is proposed to improve the current trade-off front to better trade off the accuracy of polynomial networks and the inference time under the RNS-CKKS. Intuitively, we can explore polynomials to improve the trade-off. Due to the the extremely large search space ($10^{79} \sim 10^{230}$) of the high-dimensional discrete variable $\boldsymbol{D}$ ($114 \sim 330$), random search or evolutionary algorithms for low-dimensional problems are not efficient. To efficiently solve the high-dimensional search problem, we adopt a *divide-and-conquer* strategy called **cooperative coevolution** (Ma et al., 2018; Mei et al., 2016; Yang et al., 2008), which decomposes the high-dimensional problem into multiple low-dimensional sub-problems. Every time, we can mutate the m-th EvoReLU and keep other layers fixed. To prevent local optimization, we introduce a network-level crossover so that current solutions can share *global* information.

MOCoEv: The proposed **M**ulti-**O**bjective **CoEv**olutionary search algorithm is called MOCoEv. Algorithm 1 details the implementation of the MOCoEv algorithm. MOCoEv

Algorithm 1: MOCoEv

```
Input  : Pre-trained Network $f(\boldsymbol{\omega}_0)$ including $M$ ReLUs, number of solutions $N$,
         number of sub-polynomial $K$, number of generations $T$, training dataset
         Train, mini-validation dataset Minival;
Output : Trade-off front $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \leq i \leq N$ ;
Initial : $\{\boldsymbol{D}_i\}_{i=1}^N \leftarrow$ LHS($N, M, K$) ;
         $\{\boldsymbol{\Lambda}_i\}_{i=1}^N \leftarrow$ R-CCDE($\{\boldsymbol{D}_i\}_{i=1}^N$) ;
         $\{\boldsymbol{\omega}_i\}_{i=1}^N \leftarrow$ PAT($f(\boldsymbol{\omega}_0), \{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i\}_{i=1}^N$, Train) ;
         $\{\boldsymbol{o}_i\}_{i=1}^N \leftarrow$ Eval($\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\}_{i=1}^N$, Minival) ;
         $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \leq i \leq N$ ;
for $t \leftarrow 1$ to $T$ do
    for $m \leftarrow 1$ to $M$ 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($\{\boldsymbol{D}_j, \boldsymbol{\Lambda}_j\}_{j=1}^N$) ;
        $\{\boldsymbol{D}'_j[:, m]\}_{j=1}^N \leftarrow$ Mutate($\{\boldsymbol{D}'_j[:, m]\}_{j=1}^N$) ;
        $\{\boldsymbol{\Lambda}'_j[:, m]\}_{j=1}^N \leftarrow$ R-CCDE($\{\boldsymbol{D}'_j[:, m]\}_{j=1}^N$) ;
        $\{\boldsymbol{\omega}'_j\}_{j=1}^N \leftarrow$ PAT($f(\boldsymbol{\omega}_0), \{\boldsymbol{D}'_j, \boldsymbol{\Lambda}'_j\}_{j=1}^N$, Train) ;
        $\{\boldsymbol{o}'_j\}_{j=1}^N \leftarrow$ Eval($\{\boldsymbol{D}'_j, \boldsymbol{\Lambda}'_j, \boldsymbol{\omega}'_j\}_{j=1}^N$, Minival) ;
        $\left\{\boldsymbol{D}'_j, \boldsymbol{\Lambda}'_j, \boldsymbol{\omega}'_j\right\} \Rightarrow \boldsymbol{o}'_j, 1 \leq j \leq N$ ;
        $\{\boldsymbol{o}_i\}_{i=1}^N \leftarrow$ Pareto($\{\boldsymbol{o}_i\}_{i=1}^N \cup \{\boldsymbol{o}'_j\}_{j=1}^N$) ;
        $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\} \Rightarrow \boldsymbol{o}_i, 1 \leq i \leq N$ ;
```

takes as input a neural network $f(\boldsymbol{\omega}_0)$ with M ReLUs that will be replaced by EvoReLUs. We set the number of solutions to N, the number of sub-functions of a composite polynomial to K, and the number of generations to T. We randomly select a small subset from a training dataset as the mini-validation dataset (Tan and Le, 2021). The training dataset is used to fine-tune polynomial networks, and the mini-validation dataset is used to estimate the accuracy of the polynomial networks.

In the t-th generation, it is composed of M iterations. In iteration $m, 1 \leq m \leq M$, we explore a better trade-off front by exploring the m-th layer EvoReLU and freezing other layer's EvoReLU. It is inspired by coevolution that we improve the trade-off front by solving the the m-th layer EvoReLU optimization and keeping other layers fixed. Assuming the current trade-off front $\{\boldsymbol{D}_i, \boldsymbol{\Lambda}_i, \boldsymbol{\omega}_i\}_{i=1} \Rightarrow \{\boldsymbol{o}_i\}_{i=1}, 1 \leq i \leq N$, one iteration includes the following steps (as shown in Figure 5):

① **Selection:** We randomly select N solutions from the current trade-off front to build a mating set $\{\boldsymbol{D}_j, \boldsymbol{\Lambda}_j\}_{j=1}^N$.

② **Crossover** enables network-level information exchange. As shown in Figure 5, every two solutions are combined into two new solutions. Given two solutions $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$, we mate them to obtain $\boldsymbol{D}'_1$ and $\boldsymbol{D}'_2$ where $\boldsymbol{D}'_1 = \{\boldsymbol{b}_m : \boldsymbol{b}_m \in \boldsymbol{D}_1 \cup \boldsymbol{D}_2, 1 \leq m \leq M\}$,

Algorithm 2: R-CCDE

```
Input  : Composite polynomial $p^d(x) = (p_K^{d_K} \circ p_{k-1}^{d_{k-1}} \circ \cdots \circ p_1^{d_1})(x)$ 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
        $\boldsymbol{\alpha}_k^\star \leftarrow \arg\min_{\boldsymbol{\alpha}_k} \mathcal{L}_{p^d,q}(\boldsymbol{\alpha}_k | \boldsymbol{\lambda}^*)$   s.t.   $\boldsymbol{\alpha}_k | \boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
        $\boldsymbol{\lambda}^* \leftarrow (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_k^\star, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
        $\beta_k^\star \leftarrow \arg\min_{\beta_k} \mathcal{L}_{p^d,q}(\beta_k | \boldsymbol{\lambda}^*) + \gamma \cdot \beta_k{}^2$   s.t.   $\beta_k | \boldsymbol{\lambda}^* =$
        $(\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \beta_k, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
        $\boldsymbol{\lambda}^* \leftarrow (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \beta_k^\star, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$;
```

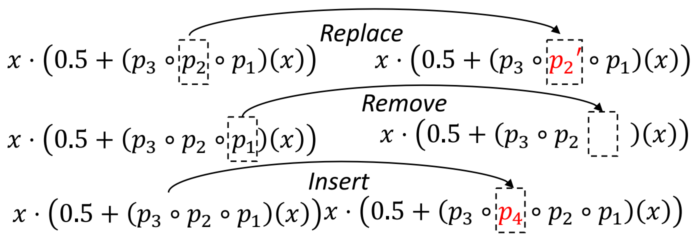
Replace
Remove
Insert
$x \cdot (0.5 + (p_3 \circ p_2 \circ p_1)(x))$
$x \cdot (0.5 + (p_3 \circ p_2' \circ p_1)(x))$
$x \cdot (0.5 + (p_3 \circ p_2 \circ \ )(x))$
$x \cdot (0.5 + (p_3 \circ p_4 \circ p_2 \circ p_1)(x))$

Figure 6: Composite polynomial mutation. Top row: randomly replace a sub-polynomial; middle row: randomly remove a sub-polynomial; bottom row: randomly insert a sub-polynomial.

$\boldsymbol{D}_2' = \{\boldsymbol{b}_m : \boldsymbol{b}_m \in (\boldsymbol{D}_1 \cup \boldsymbol{D}_2)/\boldsymbol{D}_1', 1 \le m \le M\}$. $\boldsymbol{D}_1'$ and $\boldsymbol{D}_2'$ can inherit corresponding coefficients from $\boldsymbol{\Lambda}_1$ and $\boldsymbol{\Lambda}_2$ because we exchange polynomials of the same layer.

③ **Mutation** is a coevolutionary operation. We only explore the m-th layer EvoReLU's and fix other layers. We design three types of operators to mutate a composite polynomial function. i) randomly *replace* one polynomial sub-function with a new polynomial. ii) randomly *remove* a sub-function. iii) randomly *insert* a new polynomial. Figure 6 shows how to mutate EvoReLU$(x) = x(0.5 + (p_3 \circ p_2 \circ p_1)(x))$ to generate new EvoReLU. We can randomly choose a sub-polynomial p_2 and replace it with a new sub-polynomial p_2'. We can randomly remove a sub-polynomial, like p_1, or randomly insert a sub-polynomial, like p_4;

④ **R-CCDE** optimizes polynomial coefficients.

⑤ **PAT** fine-tunes new solutions $\{\boldsymbol{D}_j', \boldsymbol{\Lambda}_j'\}_{j=1}^{N}$.

⑥ **Evaluation:** We can count the number of bootstrapping and evaluate $\{\boldsymbol{D}'_j, \boldsymbol{\Lambda}'_j, \boldsymbol{\omega}'_j\}_{j=1}^{N}$ on the minival dataset to estimate accuracy. We have a set of new solutions $\left\{\boldsymbol{D}'_j, \boldsymbol{\Lambda}'_j, \boldsymbol{\omega}'_j\right\} \Rightarrow \boldsymbol{o}'_j, 1 \leq j \leq N$.

⑦ **Update:** We obtain a new trade-off front from $\{\boldsymbol{o}_i\}_{i=1}^{N} \cup \{\boldsymbol{o}'_j\}_{j=1}^{N}$. We apply *nondominated sorting* (Deb et al., 2002) to find trade-off fronts. These $2N$ solutions will be categorized into multiple trade-off fronts. We retrain the first N solutions.

We repeat steps ①→⑦ until $m = M$, which is one generation in MOCoEv. When initializing solutions, we randomly sample $\{\boldsymbol{D}_i\}_{i=1}^{N}$ using the Latin hypercube sampling (LHS) method.

3.3.2 R-CCDE

Motivation: The EvoReLU in Equation 6 is a composite polynomial: $y_1 = p_1^{d_1}(x|\boldsymbol{\alpha}_1, \beta_1), y_2 = p_2^{d_2}(y_1|\boldsymbol{\alpha}_2, \beta_2), \cdots, y = p_K^{d_K}(y_{K-1}|\boldsymbol{\alpha}_K, \beta_K)$. The forward architecture of composite polynomials, $x \mapsto y_1 \mapsto y_2 \cdots \mapsto y_{K-1} \mapsto y$ is suitable for *coevolution* 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 $\boldsymbol{\lambda} = (\boldsymbol{\alpha}_1, \beta_1, \cdots, \boldsymbol{\alpha}_K, \beta_K)$ is grouped into $\{\boldsymbol{\alpha}_1\}, \{\beta_1\}, \cdots, \{\boldsymbol{\alpha}_K\}, \{\beta_K\}$. The coefficient $\boldsymbol{\alpha}$ controls the shape of the sub-polynomial output, while the scaling parameter β controls the amplitude. We sequentially update $\{\boldsymbol{\alpha}_k\}$ followed by β_k, $1 \leq k \leq K$. Because i) sub-polynomials close to input will greatly affect the output; ii) it is easier to learn coefficients by decoupling the amplitude from the coefficients.
Differentiable Evolution: The EvoReLU variables $\{\boldsymbol{\alpha}_k\}_{k=1}^{N}$ and $\{\beta_k\}_{k=1}^{N}$ are in the continuous space. We adopt a *simple yet effective* search algorithm to optimize these variables. Differentiable evolution (DE) (Rauf et al., 2021) by only using 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:

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

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 bits 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 initialization sensitive.

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 $p^d(x) = (p_K^{d_K} \circ p_{k-1}^{d_{k-1}} \circ \cdots \circ p_1^{d_1})(x)$ 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 $y = \text{EvoReLU}(x) = x \cdot \big(0.5 + p^d(x)\big)$ in Equation 6, we set the target function $q(x) = 0.5 \cdot \text{sgn}(x)$. We set the number of generations and the scaling decay parameter to T and γ, respectively. The objective function $\mathcal{L}_{p^d,q}(\cdot)$ is the ℓ_1 distance between the composite polynomial $p^d(x)$ and the target function $q(x)$. R-CCDE maintains a **context vector** (Mei et al., 2016) $\boldsymbol{\lambda}^* = (\boldsymbol{\alpha}_1^*, \beta_1^*, \cdots, \boldsymbol{\alpha}_K^*, \beta_K^*)$ 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

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

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 $\mathcal{L}_{p^d,q}\left(\boldsymbol{\alpha}_1^*, \cdots, \boldsymbol{\alpha}_k, \cdots, \beta_K^*\right)$. 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 is 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

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

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 $p_k^{d_k}(x) = \frac{1}{\beta_k} \sum_{i=1}^{d_k} \alpha_i \text{T}_i(x)$, 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 pushes β_k to 1. Therefore, DE is not biased toward solutions with large β_k numbers.

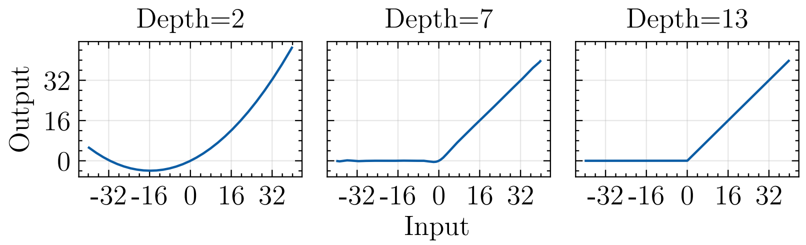

Figure 7: The 41-st EvoReLU of ResNet-56 backbone on CIFAR-10 with $B = 39.74$. Left: depth=2 corresponding to 25-bootstrapping solution; middle: depth=7 corresponding to 39-bootstrapping solution; right: depth=13 corresponding to a 47-bootstrapping solution.

3.3.3 PAT

Motivation: Replacing ReLU with EvoReLU in pre-trained 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) to adapt to the approximation error. However, backpropagation through EvoReLU easily leads to exploding gradients. Because backpropagating gradients may be exponentially amplified due to a lot of composite polynomials. We note R-CCDE can precisely evolve coefficients of EvoReLU in function level. In Figure 7, we show curves of the 41-st EvoReLU of ResNet-56 backbone on the CIFAR-10 dataset. We show three precision solutions corresponding to 25, 39, and 47 bootstrapping operations. These EvoReLU's consumes 2, 7, and 13 levels. The 7-level and 13-level solutions can precisely approximate ReLU, while the 2-level solution is a quadratic function and *relatively* precisely approximates ReLU. If we use ReLU for backpropagation, the gradient error is small, and we can avoid exploding gradients.
PAT: Thanks to the precise *forward* approximation of EvoReLU, we can use gradients from the original non-arithmetic ReLU function for *backpropagation*, specifically, during *the forward* pass, EvoReLU injects slight errors, which are captured by objective functions like Cross-Entropy loss. During the *backward* pass, we bypass 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 (Bengio et al., 2013) and QAT (Jacob et al., 2018), which uses two different functions for forward- and back-propagation. Given a polynomial network with EvoReLU $\{\boldsymbol{D}, \boldsymbol{\Lambda}\}$, we obtain polynomial-aware weight by $\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \mathcal{L}_{train}\left(f(\boldsymbol{\omega}); \boldsymbol{D}, \boldsymbol{\Lambda}(\boldsymbol{D})\right)$ in Equation 7. We inherit pre-trained $\boldsymbol{\omega}_0$ from the ReLU-based network to initialize $\boldsymbol{\omega}$. $\mathcal{L}_{train}$ is Cross-Entropy loss on the training dataset. We fine-tune $\boldsymbol{\omega}$ for **a few epochs** to get $\boldsymbol{\omega}^*$ compatible with layerwise EvoReLU's. The following pseudocode illustrates this procedure for a simple example, $\text{EvoReLU}(x) = x(0.5 + (p_3 \circ p_2 \circ p_1)(x))$. In the forward function, we first scale the coefficients of p_3 by B so that the output range of y is $[0, B]$.

```
def forward($x$, $B$):
    $p_3 = p_3 \cdot B$
    $y = p_3(p_2(p_1(x)))$
    $y = x(0.5 + y)$
    return $y$
```

In the backward function, we compute the gradient $\partial y/\partial\text{ReLU}(\text{x})$ instead of $\partial y/\partial\text{EvoReLU}(\text{x})$ to avoid exploding gradients.

```
def backward(x, grad):
    y = ReLU(x)
    d_y = ∂y/∂x
    grad = grad · d_y
    return grad
```

4. Experiments

Setup: We benchmark AutoFHE on CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009). Both datasets have 50,000 training and 10,000 validation images at a resolution of 32×32. CIFAR-10 has 10 classes, while CIFAR-100 includes 100 classes. The validation images are treated as private data and used only for evaluating the final networks. We randomly select 5,120 images from the training split as a *minival* (Tan and Le, 2021) dataset to guide the search process. The Top-1 accuracy on the minival dataset optimizes 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*. To evaluate AutoFHE under RNS-CKKS, we adopt the publicly available code of FHE-MP-CNN and adapt it for inference with layerwise EvoReLU. During inference, we keep track of the ciphertext levels and call the bootstrapping operation when the level reaches zero, thanks to the optimal placement of bootstrapping operations found by AutoFHE. For a fair comparison between AutoFHE and the baseline FHE-MP-CNN, we use the pre-trained network weights provided by FHE-MP-CNN.
Hyperparameters: For MOCoEv, we use a set of 50 solutions and run it for 20 generations. For mutation of MOCoEv, we replace a sub-polynomial, remove a sub-polynomial and insert a sub-polynomial with probabilities 0.5, 0.4 and 0.1, respectively. 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 $10\times$ the number of variables. We set the scaling decay to $\gamma = 0.01$ and the number of iterations to 200. For PAT, we use a batch size of 512 and weight decay of 5×10^{-3} and clip the gradients to 0.5. We use learning rates of 5×10^{-4} for CIFAR-10 and 2×10^{-4} for CIFAR-100. During MOCoEV search, we set the fine-tuning epoch to one. After the search is done, we fine-tune searched polynomial networks for ten epochs. On one NVIDIA RTX A6000 GPU, the search process for ResNet-20/32/44/56 on CIFAR-10 took 59 hours, 126 hours, 200 hours, 281 hours, respectively. The search for ResNet-32 on CIFAR-100 took 140 hours.

4.1 AutoFHE Trade-Off Fronts

Figure 9 shows trade-off fronts found by AutoFHE on CIFAR-10 and CIFAR-100 for different ResNet models. The trade-offs are between the Top-1 validation accuracy on plaintext validation datasets and the number of bootstrapping operations required for the corresponding homomorphic evaluation architecture. Please note we report accuracy on plaintext data to show *overall* trade-off fronts discovered by AutoFHE. By optimizing the bi-level multi-objective in Equation 7, AutoFHE adapts to the differing sensitivity of the activation layers to approximation errors and reduces the number of levels required compared to using the

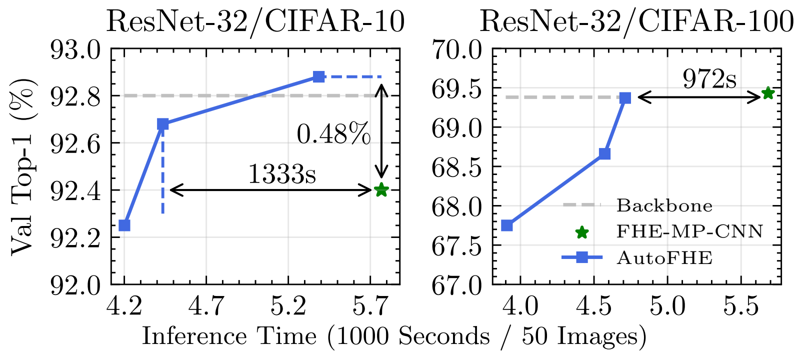

Figure 8: Trade-offs of AutoFHE for ResNet-32 on CIFAR-10/-100 under RNS-CKKS.

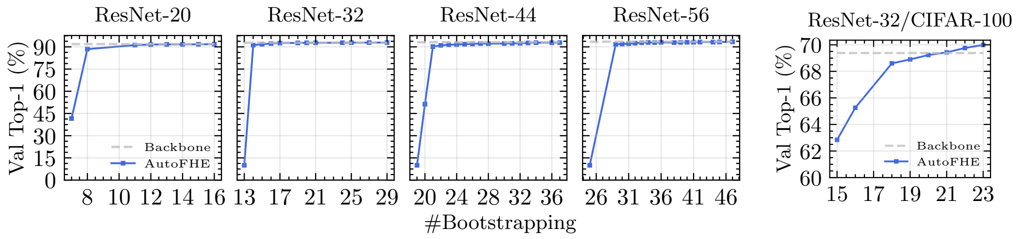

Figure 9: AutoFHE Pareto trade-off fronts on ResNet backbones. We report the accuracy on plaintext validation datasets and the number of bootstrapping operations. Left: ResNet-20/32/44/56 on CIFAR-10; Right: ResNet-32 on CIFAR-100.

same high-degree AppReLU in all the layers. Thus, AutoFHE significantly reduces the number of bootstrapping operations. For ResNet-32 on CIFAR-10, AutoFHE removes 10 bootstrapping operations (33.33%) compared to FHE-MP-CNN with the negligible accuracy loss 0.08% compared to the original network with ReLUs. Lastly, AutoFHE provides a family of solutions offering different trade-offs rather than a single solution, thus providing flexible kinds of service to meet different requirements.

4.2 AutoFHE under RNS-CKKS

Due to the high computation cost of validating networks performance on encrypted data under the RNS-CKKS, we select **nine solutions** for evaluation on a machine with AMD EPYC 7H12 64-Core Processor and 1000 GB RAM. In Table 3, we evaluate three solutions for ResNet-32 on both CIFAR-10 and CIFAR-100 and evaluate one solution for ResNet-20/-44/-56. We estimate the inference time for 50 images on 50 CPU threads. Amortized inference time is amortized runtime for each image. We report the Top-1 accuracy of AutoFHE on all (**10,000**) encrypted validation images under the RNS-CKKS. We plot trade-off fronts on CIFAR-10 of AutoFHE versus FHE-MP-CNN in Figure 4. Figure 8 shows trade-off fronts of AutoFHE of ResNet-32 on encrypted CIFAR-10/-100.

We observe that, on CIFAR-10, AutoFHE provides significant acceleration while having better accuracy or preserving accuracy. AutoFHE of ResNet-32 with 21 bootstrapping operations has slightly better accuracy than the baseline of ResNet-44 and accelerates

Dataset	Backbone		FHE-MP-CNN				**AutoFHE**			
	Network	Top-1	Boot	Top-1*(%)	Inference	Amortized	Boot	Top-1(%)	Inference	Amortized
CIFAR-10	ResNet-20	91.86	18	91.31	3,532s	71s	**13**	**91.39**	**2,643s**	**53s**
	ResNet-32	92.80	30	92.40	5,768s	115s	**20**	92.25	**4,201s**	**84s**
							21	92.68	4,435s	89s
							29	**92.88**	5,386s	108s
	ResNet-44	93.13	42	92.65	7,732s	155s	**38**	92.04	**7,209s**	**144s**
	ResNet-56	93.49	54	93.07	9,837s	197s	**47**	**93.27**	**8,684s**	**174s**
CIFAR-100	ResNet-32	69.38	30	**69.43**	5,684s	114s	**21**	67.75	**3,908s**	**78s**
							22	68.66	4,573s	91s
							23	69.37	4,712s	94s

Table 3: AutoFHE under the RNS-CKKS scheme. Top-1* accuracy for FHE-MP-CNN, as reported in (Lee et al., 2022a). The inference time for 50 images is evaluated on AMD EPYC 7H12 64-core processor using 50 threads. Boldface denotes the best criterion on a backbone network, like the best Top-1 accuracy and the least inference time; underline denotes AutoFHE outperforms FHE-MP-CNN. AutoFHE even discovers two polynomial CNNs of ResNet-32 showing better accuracy compared with FHE-MP-CNN's ResNet-44 solution.

Dataset	Backbone	FHE-MP-CNN				**AutoFHE**			
		Boot	AppReLU	Bootstrapping	Inference	Boot	EvoReLU	Bootstrapping	Inference
CIFAR-10	ResNet-20	18	331 ± 13s	$2,651 \pm 29$s	$3,450 \pm 40$s	13	208 ± 4s	**1,767** ± 10s	$2,626 \pm 9$s
	ResNet-32	30	537 ± 16s	$4,309 \pm 69$s	$5,577 \pm 96$s	20	273 ± 5s	**2,731** ± 34s	$4,136 \pm 42$s
						21	293 ± 8s	**2,907** ± 42s	$4,340 \pm 53$s
						29	380 ± 8s	**3,835** ± 41s	$5,270 \pm 59$s
	ResNet-44	42	724 ± 23s	$5,850 \pm 90$s	$7,551 \pm 117$s	38	499 ± 12s	**5,183** ± 49s	$7,112 \pm 66$s
	ResNet-56	54	929 ± 22s	$7,363 \pm 118$s	$9,525 \pm 155$s	47	642 ± 14s	**6,243** ± 68s	8543 ± 94s
CIFAR-100	ResNet-32	30	536 ± 24s	$4,236 \pm 60$s	$5,526 \pm 86$s	21	305 ± 8s	**2,566** ± 30s	$3,852 \pm 46$s
						22	338 ± 9s	**3,012** ± 57s	$4,446 \pm 69$s
						23	359 ± 9s	**3,225** ± 23s	$4,660 \pm 33$s

Table 4: Average runtime of AppReLU/EvoReLU and bootstrapping operations. We report the average runtime of operations of 50 images in Table 3. Inference time is the average runtime of 50 threads that is a slightly different from inference time in Table 3 because of multi-thread overhead. Bootstrapping operations dominate the inference latency in FHE-MP-CNN. AutoFHE effectively accelerates inference by reducing the number of bootstrapping operations.

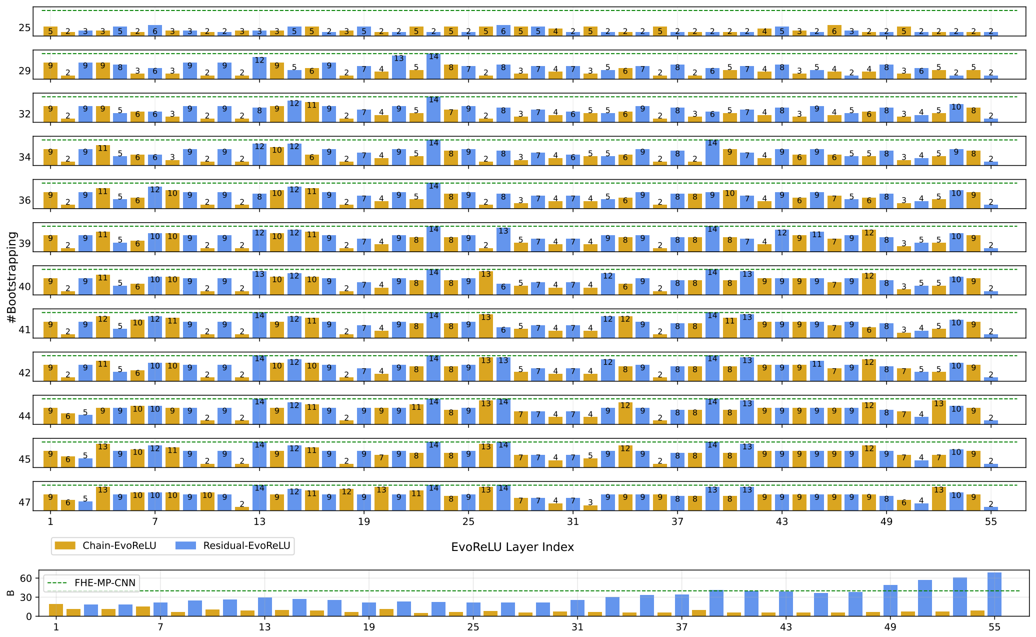

Figure 10: Depth consumption distribution of EvoReLUs of ResNet-56. Top: depth consumption distributions of layerwise EvoReLUs of solutions spanning the trade-off front with different bootstrapping consumption. Bottom: the distribution of scaling parameters (B) of layerwise EvoReLUs. The green dashed lines show the depth consumption or B of AppReLUs of FHE-MP-CNN.

inference by **3,297** seconds (**43**%). AutoFHE reduces inference time of ResNet-20, ResNet-32 (21 bootstrapping operations) and ResNet-56 by **25**%, **23**% and **12**% compared with the corresponding solutions of FHE-MP-CNN while improving accuracy up to **0.28**%. AutoFHE with 29 bootstrapping operations improves accuracy of ResNet-32 by **0.48**% while accelerating inference by 382 seconds (7%). AutoFHE can achieve a Top-1 accuracy of **91.39**% on encrypted CIFAR-10 under the RNS-CKKS at an amortized inference latency of under **one minute (53 seconds) per image**, which brings us closer towards practically realizing secure inference of deep CNNs under RNS-CKKS. On CIFAR-100, AutoFHE saves inference time by **972** seconds (**17**%) while preserving the accuracy. The experiments prove that AutoFHE can find trade-off fronts of accuracy and inference time. Furthermore, the results validate our assumption that directly reducing the number of bootstrapping operations can effectively accelerate inference speed.

We provide average runtime of different operations in FHE-MP-CNN and AutoFHE in Table 4. From Table 4, we conclude:

- Bootstrapping dominates inference time (**77%**), while high-degree AppReLUs incurs high consumption of bootstrapping operations but only consumes inference time (**10%**).

- AutoFHE can effectively accelerates CNN inference on RNS-CKKS by removing bootstrapping operations. AutoFHE also slightly benefits from evaluating cheaper EvoReLU.

- Bootstrapping still dominates inference time in AutoFHE. This will be even more prominent when considering larger networks on more challenging datasets like ImageNet (Russakovsky et al., 2015). Therefore, we believe that saving levels → saving bootstrapping is an important and an exciting direction of future research.

4.3 AutoFHE Depth Consumption

To verify the *layerwise* assumption of AutoFHE, depth consumption distribution of Layerwise EvoReLU for ResNet-56 with different bootstrapping operations are in shown Figure 10. We make three **observations** that can guide designing polynomial neural networks under FHE.

Observation 1

Activation position affects depth consumption

Residual EvoReLUs consume more levels than chain EvoReLUs, suggesting that residual ReLU layers have less tolerance to approximation errors

Observation 2

Normalization decreases depth consumption

Since pre-activations of chain EvoReLUs are normalized, they follow a tighter distribution and need smaller scaling values. We can consider using more BN layers to design FHE-friendly networks.

Observation 3

Output layer prefers low-degree polynomials

The last EvoReLU close to the output does not need high-degree polynomials. They are close to the loss function and are better suited to learn polynomial-aware weights.

5. Conclusion

This paper introduces AutoFHE, a multi-objective search system for generating polynomial networks under FHE. AutoFHE enables us to use pre-trained CNNs and provide different service to customers. AutoFHE seeks to search and learn the end-to-end function represented by the network instead of approximating each activation. We exploit layerwise polynomial activations across different layers in a network and jointly search for placement of bootstrapping operations for evaluation under RNS-CKKS. Experimental results over ResNets on CIFAR-10 and CIFAR-100 indicate that AutoFHE can reduce the inference time by up to 3,297 seconds (43%) while preserving the accuracy. AutoFHE also improves the accuracy by up to 0.48%. Although our focus in this paper is on ResNets, and consequently ReLU, AutoFHE is a general-purpose algorithm that is agnostic to the network architecture or the type of activation function.

References

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`.

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. Springer, 2021. `https://link.springer.com/chapter/10.1007/978-3-030-77870-5_21`.

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

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. Springer, 2017. `https://link.springer.com/chapter/10.1007/978-3-319-70694-8_15`.

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

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. Springer, 2018b. `https://link.springer.com/chapter/10.1007/978-3-030-10970-7_16`.

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`.

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`.

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`.

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`.

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. *arXiv preprint arXiv:2010.11929*, 2020. `https://openreview.net/pdf?id=YicbFdNTTy`.

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`.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016. `http://proceedings.mlr.press/v48/gilad-bachrach16.pdf`.

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`.

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`.

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`.

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`.

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`.

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*, 2021a. `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9517029`.

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. PMLR, 2022a. https://proceedings.mlr.press/v162/lee22e/lee22e.pdf.

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 *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*, pages 618–647. Springer, 2021b. https://eprint.iacr.org/2020/552.pdf.

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, 2022b. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9734024.

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*, 2021c. https://arxiv.org/pdf/2105.10879.pdf.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. https://arxiv.org/abs/1806.09055.

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. PMLR, 2021. http://proceedings.mlr.press/v139/lou21a/lou21a.pdf.

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-.

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.

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 (TOMS)*, 42(2):1–24, 2016. https://dl.acm.org/doi/pdf/10.1145/2791291.

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

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.

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`.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. `https://link.springer.com/article/10.1007/s11263-015-0816-y`.

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`.

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`.

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`.

Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, 2021. `http://proceedings.mlr.press/v139/tan21a/tan21a.pdf`.

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. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021. `https://proceedings.neurips.cc/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

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/`.

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`.

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

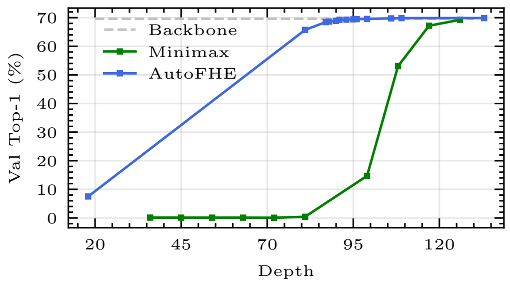

Figure 11: Evaluate AutoFHE over ResNet-18 on ImageNet.

A. AutoFHE on ImageNet

It is not practical to evaluate high-resolution images under the RNS-CKKS scheme due to the extremely high memory footprint and computational complexity constraints. We evaluate AutoFHE on plaintext ImageNe (Russakovsky et al., 2015) to demonstrate its efficacy on the large-scale high-resolution dataset. We set the number of generations to 10, the size of the minival dataset to 2,560, and the solution size to 30. Other hyper-parameters are as same as CIFAR experiments. We turn off fine-tuning during the search and fine-tune the final result for one epoch. The search experiment took 18 hours. We estimate accuracy on the plaintext validation dataset and use depth consumption as the inference cost under the RNS-CKKS. We adopt the Minimax polynomials with precision from 4 to 13 (Lee et al., 2021c) as our baseline and set $B = 100$. Figure 11 shows trade-off fronts of AutoFHE and Minimax. AutoFHE has 69.26% accuracy with a depth consumption of 91, while Minimax consumes 126 depth to have the same accuracy. AutoFHE can reduce 28% depth consumption. When depth consumption is equal to 99, Minimax has an accuracy 14.71%. AutoFHE reports accuracy 65.56% with the same depth consumption. This experiment demonstrates that AutoFHE can effectively trade off the accuracy and the depth consumption on large-scale high-resolution datasets.

B. Layerwise EvoReLUs of ResNet-56

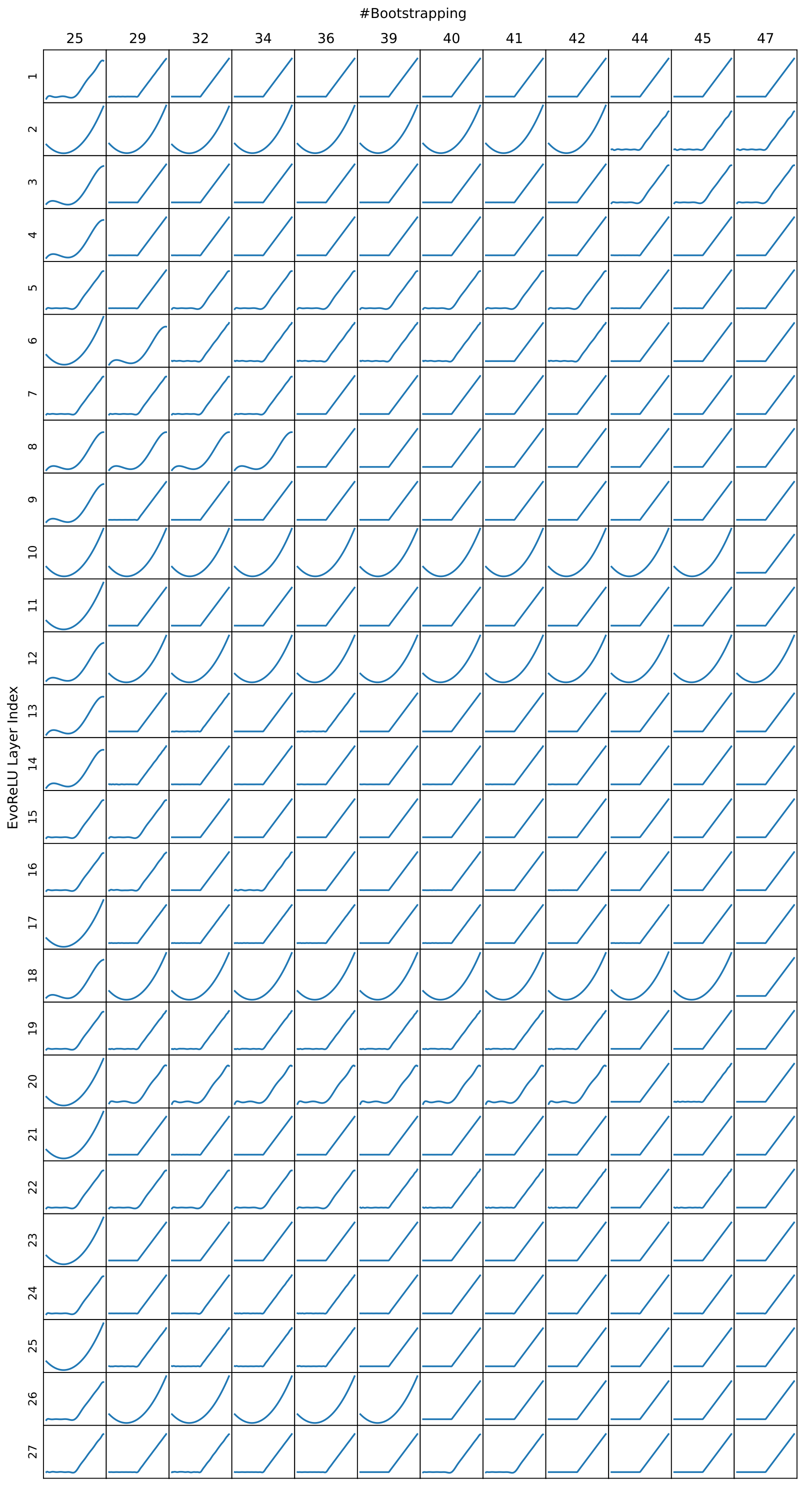

Figure 12: EvoReLUs of ResNet-56 from layer 1 to 27.

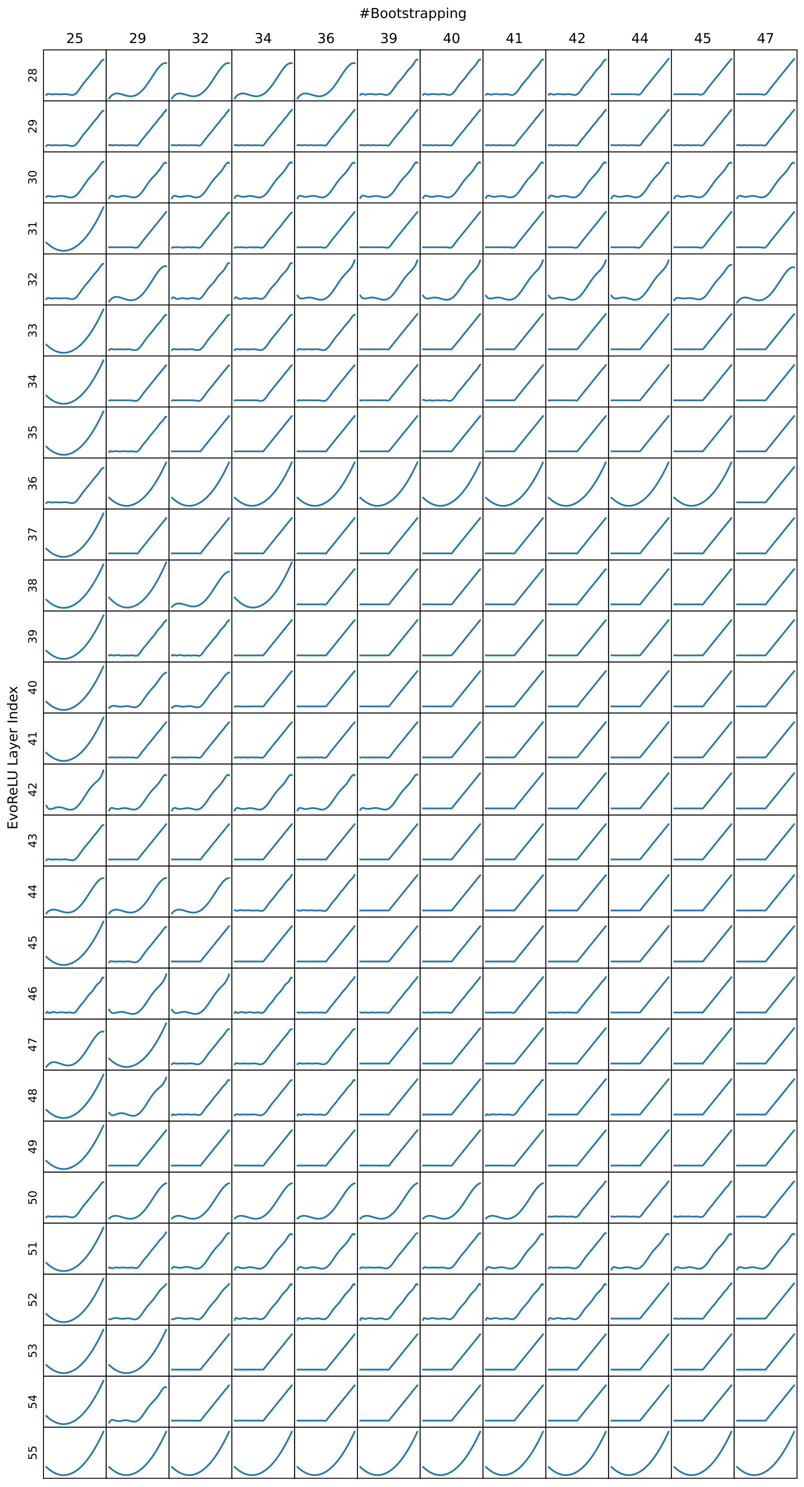

Figure 13: EvoReLUs of ResNet-56 from layer 28 to 55.