

AceHomo: Accelerating Privacy Preserving Inference through Dynamic Level Adjustment

Hongyan Li^{*1,2,3}, Jinkai Zhang^{*1,2}, Hang Lu^{1,2,3,✉}, and Xiaowei Li^{1,2,3}

¹SKLP, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

²University of Chinese Academy of Sciences, Beijing, China

³Zhongguancun Laboratory, Beijing, China

lihongyan@mail.zgclab.edu.cn, {zhangjinkai22z, luhang, lxw}@ict.ac.cn

Abstract—Fully Homomorphic Encryption (FHE) neural networks has made significant strides in enabling privacy-preserving inference, yet it is frequently impeded by substantial computational overhead. Previous research has largely focused on optimizing neural network operations or managing FHE computation processes to reduce computational cost, but has overlooked addressing the interplay between FHE computation process and neural network characteristics as an integrated system. This paper introduces AceHomo, a novel approach utilizing reinforcement learning to optimize CKKS encryption level consumptions in neural networks, thereby enhancing performance and reducing latency without compromising network accuracy. Experimental results demonstrate that AceHomo significantly reduces latency by up to 1.47x without compromising the network's inference accuracy. These encouraging results position AceHomo as a promising approach in optimizing FHE neural networks.

Index Terms—Fully homomorphic encryption, Neural networks, Reinforcement learning.

I. INTRODUCTION

The exponential growth of Artificial Intelligence (AI), fueled by deep learning and increased computational power, has transformed sectors like healthcare, finance, and social media, while raising critical privacy and security concerns due to the handling of vast amounts of sensitive data. This underscores the urgent need for effective privacy-preserving techniques in AI.

In this context, privacy-preserving neural networks have emerged as a pivotal area of research. These networks aim to maintain data confidentiality while enabling meaningful analysis and inference. A leading approach in this field is Fully Homomorphic Encryption (FHE) [1]. FHE is considered to be one of the most suitable tools for implementing privacy preserving machine learning (PPML), enabling both strong security in the cryptographic sense and simplicity of communication. FHE supports computation on encrypted data without decryption, which enables Machine Learning as a Service (MLaaS) providers to directly infer their customers' encrypted personal data. These advantages have motivated researchers to investigate FHE-based neural network.

* These authors contributed equally to this work. Corresponding author is Hang Lu. This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University.

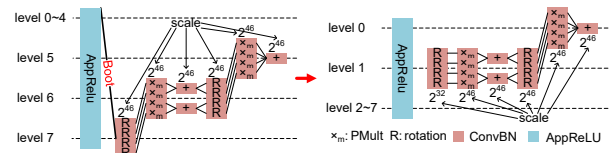


Fig. 1. Illustration of convolutional layer optimization of ResNet20 in CKKS scheme. Conv consists of an initial series of rotations, followed by a ciphertext multiplication, and concludes with another set of rotations and a subsequent multiplication. It's assumed bootstrapping can refresh the ciphertext level to 7.

Homomorphic encryption, particularly the CKKS [2], [3] scheme, represents a significant advancement in privacy-preserving technologies by enabling approximate computations on encrypted data. Due to its characteristics, the CKKS scheme is widely applied in homomorphic neural networks [4], [5]. While FHE holds significant promise for secure AI applications, its practical implementation encounters considerable challenges, particularly concerning computational efficiency and overhead. Addressing these challenges is critical for advancing the practicality and performance of privacy-preserving AI systems, making it a compelling area for ongoing research and development.

Several works have aimed to improve the efficiency of homomorphic neural networks by reducing computational overhead. Existing approaches typically adapt neural network architectures and enhance the efficiency of convolutional operations and non-linear activation functions after homomorphic encryption. However, these methods often overlook the unique interplay between neural network robustness and FHE computation, which is crucial for privacy-preserving inference. Specifically, the potential for leveraging neural networks' fault tolerance to manage errors within FHE remains underexplored. Figure 1 illustrates the interplay between neural network dynamics and the CKKS computation process, particularly in convolution operations. By strategically adjusting the level at which convolutional layers operate, we can eliminate the need for bootstrapping operations [6], [7], leading to a significant reduction in latency. This adjustment, however, may cause a minor increase in output error due to the initial scale reduction to 2^{32} . This highlights the underutilization of neural networks' fault tolerance in FHE, suggesting that the full potential of FHE neural networks has not yet been realized.

To bridge the gap, we introduce AceHomo, an optimization framework employing reinforcement learning (RL) to dynamically adjust CKKS computation level consumption (detailed in Section III-A). Our approach considers the complex interplay between accuracy, and the computational overhead of homomorphic networks. By optimizing the interplay, we aim to minimize latency and error in CKKS neural network inference, thus offering a more efficient and automated solution that addresses key challenges in the field.

The contributions of this paper are as follows: **Development of an RL-Based Optimization Framework:** We propose a novel framework, AceHomo, that uses RL to dynamically adjust FHE levels consumption. This framework considers the complex interplay between the runtime performance and accuracy of homomorphic network models. **Experimental Evaluation:** We conduct extensive experiments to evaluate the effectiveness of the AceHomo method. Our results demonstrate that AceHomo can significantly improve the efficiency of encrypted neural network inference, achieving up to 1.47x reduction in latency with minimal impact on accuracy. In summary, this paper presents a significant advancement in the optimization for encrypted deep learning inference.

II. BACKGROUND AND MOTIVATION

A. RNS-CKKS Fully Homomorphic Encryption

RNS-CKKS [3] is a fully homomorphic encryption scheme that supports fixed-point arithmetic and utilizes the Residue Number System (RNS) to enhance computational efficiency. RNS-CKKS encodes a vector of $N/2$ complex numbers into a plaintext of a cyclotomic polynomial of degree N , which is then encrypted into a ciphertext consisting of two polynomials. Each ciphertext has a precision attribute, scale, and a modulus limit, Q . Decryption fails if scale exceeds Q due to noise overflow. The coefficient modulus Q is composed of multiple small moduli arranged in a modulus chain q . The number of remaining moduli is denoted as level, ranging from 0 (indicating only one modulus remaining on the modulus chain) to L_{max} (indicating no consumption of moduli). The maximum modulus limit Q_{max} is constrained by N .

Homomorphic multiplication in RNS-CKKS increases the scale of the ciphertext. To maintain the scale and level of the ciphertext, four operators are introduced: Rescale, Upscale, Modswitch, and Bootstrapping. Rescale consumes the last modulus in the chain, reducing the scale by the corresponding modulus value. Upscale increases scale by multiplying a specific scale. Modswitch consumes the last modulus to reduce the level of the ciphertext. Bootstrapping refresh the ciphertext level to L_{Boot} and scale to S_f . It requires a specialized modulus chain structure with two different types of moduli: S_f and S_b .

RNS-CKKS introduces errors during computation, which accumulate as operations proceed. To ensure that these errors do not affect the final computation results, the scale of all ciphertexts must not fall below a predefined value, denoted as the waterline according to [8]. The waterline is variable and can be adjusted at different stages of the program, as proposed in ELASM [9].

B. Gap Between Neural Networks and CKKS Optimization

The advent of fully homomorphic neural networks represents a significant advancement in privacy-preserving inference. However, their performance is often hindered by the substantial computational overhead inherent in encryption mechanisms. Core computations, such as convolution and activation functions, incur significant penalties when conducted on ciphertexts.

Convolution, a core operation in CNN, entails the sliding multiplication of an image vector with a convolution kernel. GAZELLE [10] has pioneered an efficient single-input single-output channel convolution (SISO) mechanism by leveraging the SIMD technique. [11] introduced a multiplexed convolution algorithm capable of handling convolutions for multiplexed input tensors. Nonlinear activation functions, exemplified by the ReLU function, are incompatible with direct application in FHE and must be approximated [12]–[14] through operations supported by FHE. To sustain a low approximation error across a broad range and to ensure the fidelity of the CNN model, it is customary to augment the number of fitted polynomials. This approach, however, engenders considerable runtime overhead.

Beyond post-encryption operation optimization, research has also been directed towards managing FHE computation process to mitigate overhead [9], [15]–[17]. For example, Dacapo [16] and Phelipe [17] optimize bootstrapping placement to reduce latency. ELASM [9] introduces a dynamic waterline approach, allowing each operator's waterline to be tailored to its specific needs. By incorporating a new scale-to-noise ratio (SNR) parameter, ELASM identifies more efficient strategies for Rescale and Modswitch operations.

Despite significant advancements, current methodologies lack an integrated approach that treats FHE computation and neural networks as a cohesive system. Instead, they either retrofit neural networks to enhance post-encryption efficiency without considering the broader impact of FHE parameters or focus solely on optimizing FHE computation, neglecting the robustness of the neural network itself. However, a co-optimization strategy that harmonizes both aspects can yield substantial performance improvements, as shown in Figure 1.

Neural networks' inherent fault tolerance allows them to endure computational errors without compromising inference accuracy. This robustness has been leveraged in optimizing plaintext neural networks [18], [19] accelerating computational processes while preserving accuracy. The CKKS encryption scheme, which allows for computational approximations, provides a novel opportunity for optimization. By aligning CKKS approximations with neural networks' fault tolerance, we can enhance encrypted computation efficiency without sacrificing accuracy. This co-optimization strategy offers a promising path for refining CKKS to optimize neural network computations.

C. Opportunity for Optimizing CKKS-Encrypted Networks

However, harnessing the co-optimization is challenging. Firstly, the fine-grained operation granularity of FHE does not align with the computational graph structure of encrypted neural networks, which typically involve a multitude of operators. Secondly, the parameters in CKKS computation process

results in a huge search space, complicating the optimization process. Finally, aside from the parameter values themselves, bootstrapping insertion, which is intimately linked to latency and accuracy, should also be considered in the optimization.

To address optimization granularity, we propose the cell granularity. The approach distinguishes itself from the fine-grained operations of CKKS and the broader granularity of entire network layers. At this cell granularity, optimizations such as bootstrapping insertion and waterline management are performed without examining each cell's internal operations. This approach segments layers based on their multiplication depth: Layers with higher depth, such as ReLU, are divided into multiple cells. The ReLU function, for instance, is approximated using three nested polynomials to fit the $\text{sign}(x)$ function and is ultimately represented as $0.5 * (\text{sign}(x) + 1)$ [20]. In contrast, layers with lower depth, such as convolutional, downsampling, or fully connected layers, are represented by a single cell. Bootstrapping insertion strategies are adapted from prior research.

In optimizing CKKS computations for neural networks, parameters such as computation levels, waterline, and scaling factors are critical for determining runtime performance and error, which are key indicators of accuracy in privacy-preserving inference. The latency of homomorphic network operations is closely tied to the frequency of bootstrapping operations, and the error magnitude is directly proportional to the waterline. Notably, the levels parameter significantly impact both bootstrapping necessity and waterline determination, affecting overall efficiency and accuracy. Consequently, we concentrate our optimization efforts on the level parameters, tackling the vast search space mentioned earlier.

Figure 2 illustrates the interplay between CKKS encryption levels, latency, and error in CKKS-encrypted neural networks. Each subplot maps CKKS level combinations (x-axis) against latency (y-axis), with error magnitude visualized through point size/color. We adjust levels in random cells while keeping other operations constant. The case₀ highlights optimal level combinations balancing low latency and acceptable error. In Figure 2, we regard a configuration of level parameters to be advantageous if it yields reduced lower latency accompanied by an acceptable magnitude of error, as opposed to merely pursuing the minimization of error. We hold this view based on the premise that the inherent robustness of neural networks can tolerate the accuracy degradation associated with these errors, thereby boosting overall network performance.

The synergistic utilization of neural network fault tolerance and the approximate nature of CKKS encryption is a promising opportunity for optimization. By meticulously adjusting the CKKS computational level parameters to align with the fault tolerance of neural networks, we can enhance the efficiency of encrypted computations without sacrificing neural network accuracy.

D. Multi-Objective and Asymmetric Optimization

The optimization of CKKS neural networks is a complex endeavor, involving the intricate interplay of multiple level

parameters within the CKKS computation process. These parameters significantly affect the latency and error integral to privacy-preserving inference tasks. The selection of CKKS levels is fraught with long-term dependencies, posing a multifaceted optimization challenge. The choice of a single level parameter can profoundly affect others and the bootstrapping strategy across the CKKS system, adding to the complexity of the optimization process. Balancing the minimization of latency with the preservation of network accuracy is essential.

Our goal is to significantly reduce the latency in CKKS neural networks without compromising accuracy. This presents an asymmetric multi-objective optimization problem, prioritizing latency reduction while ensuring accuracy remains within acceptable limits. Reinforcement learning (RL) is particularly suited for this task due to its capacity to learn optimal strategies through dynamic interactions with the environment, which is ideal for scenarios demanding continuous decision-making and long-term planning. In this paper, we apply RL to dynamically adjust strategies in response to feedback, focusing on the latency and error of encrypted networks. Our aim is to fine-tune the CKKS computation process for encrypted deep learning, automating the parameter adjustment to find the balance between accuracy and computational efficiency.

In conclusion, while FHE provides robust privacy-preserving capabilities, the computation overhead presents a significant challenge. RL offers a promising path forward by addressing the intricate interplay between CKKS computation process and performance of network. By leveraging RL, we can effectively address the complex, dynamic nature of CKKS optimization, steering towards more efficient and accurate encrypted network.

III. METHODOLOGY

A. Methodology Overview

Our methodology, designated as AceHomo, introduces a RL framework designed to optimize CKKS computation levels across individual cells within a neural network. It aims to reduce latency while keeping encryption errors within acceptable limits. As discussed in Section II, the CKKS level parameters adjustment offers a promising space for optimization. However, optimizing the specific level numbers in the computation process can significantly expand the search space and amplify the complexity of optimization due to the vast number of potential levels.

To simplify this, we introduce the concept of **level consumption**, which refers to the computational level required to complete a cell's computation. A core insight of our approach is that reducing level consumption reduces the frequency of bootstrapping operations, leading to lower latency. This insight is central to our strategy: by minimizing level consumption, we can decrease the need for bootstrapping operations, thereby reducing latency. Therefore, for each computational cell, optimizing level consumption is a key focus within AceHomo.

AceHomo navigates the intricate relationship between latency and CKKS computation error by meticulously designing its state space to include both the network's CKKS cell parameters and dynamic encryption states, as detailed in Section III-B.

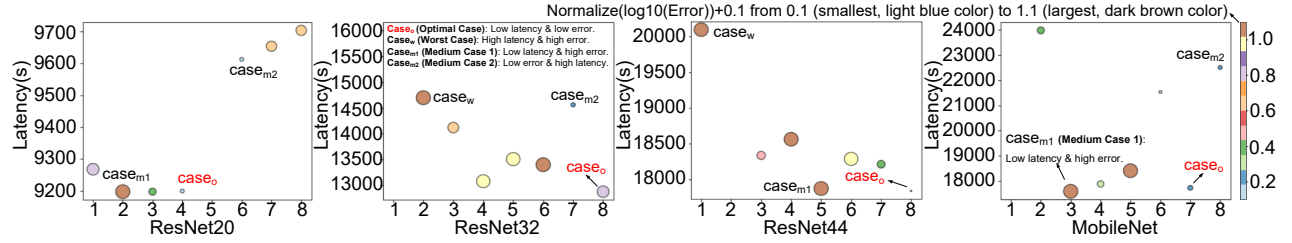


Fig. 2. Scatter Plots of varying CKKS Level Parameters vs. Latency and Error. The subplot's x-axis represents the combination of varying CKKS level parameters. Notable points with lower latency and error are marked with a case_o, indicating ideal scenario with efficient performance and high accuracy.

TABLE I
STATE SPACE INFORMATION OF ACEHOMO CELL

Cell Type	Conv/FC	ReLU
Network Specific State	L1-Norm Of Weight	Coefficients
	Input Size	Of The
	Output Size	Approximated
	Weight Parameter Size	Function
	Index Of Operator Cell Source	
Dynamic State	Optimized + Unoptimized Cell Levels	

The RL agent systematically assigns level consumption to each cell, and once the level consumption is determined, the CKKS computation management environment allocates the appropriate level for each operation and the optimal bootstrapping insertion scheme. This process estimates latency, while error management minimizes inference errors. As encrypted network inference is time-consuming, AceHomo relies on these estimations as environmental feedback. The agent receives a reward reflecting the balance between latency and computation error, as explained in Section III-B, enabling dynamic adjustments to CKKS level consumptions for optimized encrypted neural network inference performance. The REINFORCE algorithm is employed to train the AceHomo agent. Further details on its components are provided in subsequent sections.

B. Reinforcement Learning Components

The AceHomo RL model has been meticulously designed to optimize CKKS encryption within a target network as shown in Figure 3. This network is systematically divided into cells, which act as the core and minimum building blocks of the acceleration framework.

State Space: As the fundamental building block of the framework is the cell, The state space in our reinforcement learning framework is intricately designed to reflect the unique attributes of each cell type within the network. This design is crucial for the RL agent to make informed decisions that optimize the encryption process within the CKKS scheme. As shown in Table I, the state representation is bifurcated into static network state and dynamic state, which emerge from interactions with the environment. The following provides a detailed explanation of the states associated with different cells.

For convolutional and fully connected cells, the state representation includes L1-Norm of the weights. This metric is pivotal as it correlates with the magnitude of the encrypted error

generated during the multiplication of ciphertext inputs with plaintext weights, directly impacting the accuracy of CKKS convolution operations. Additionally, the state information encompasses the input size, weight size, and output feature map dimensions, which are essential for gauging the computational complexity of ciphertext addition and rotation operations within the CKKS scheme [10], [11], [21], [22].

As previously mentioned, the ReLU function is divided into four cells based on the individual functions they employ. Specifically, each cell is associated with a distinct function, and the polynomial coefficients of these functions serve as the state representation.

Furthermore, within the CKKS computation process, the level of a particular operator is influenced by the levels of other operators involved. To enrich the state information and provide a more nuanced understanding of the encryption, we incorporate the index of the source element feeding into the current cell's input. This additional information is concatenated to the aforementioned state information, thereby enhancing the state representation.

As we utilize a reinforcement learning agent to sequentially learn the level consumption of each cell, it is essential to enrich our state representation with dynamic information that reflects the interaction with the environment. Beyond the static state representation, we include the level status of each cell post-interactions with the environment. This dynamic state is represented by the optimized level consumptions and the level consumptions pending optimization across all cells. For instance, with n cells in the network, the initial dynamic level state of the first cell comprises the unoptimized level consumptions of all n cells. Subsequent cells' dynamic level states reflect the optimized consumption of preceding cells and the remaining unoptimized consumptions, ensuring a comprehensive understanding of the encryption state within the CKKS scheme.

Action Space: The initial level consumptions for unoptimized cells set the upper boundary of our agent's action spaces. To reduce these consumptions, thereby defining our action space. The action space is comprised of a finite set of possible level consumptions, such as $\{2, 3, 4, 5\}$, which correspond to the potential level consumption choices for each cell. To accommodate the variations in selection space lengths among different cells, we utilize a masking mechanism. This technique selectively disables certain actions, ensuring that the action space for each cell aligns with its unique encryption level

requirements.

Policy Network: As depicted in Figure 3, in the design of our RL policy network, we employ Gated Recurrent Unit (GRU) unit to encapsulate both the network-specific and dynamic state information of each cell following its interactions with the environment. The input vector fed into the GRU for each cell is composed of the cell's static state attributes, while the hidden state input is derived from the dynamic transitions in cell level consumption, which incorporates the optimized consumption levels of antecedent cells. The output of the GRU unit, after passing through a fully connected layer and a SoftMax activation function, determines the optimized level consumption for the current cell within the action space. The execution of the GRU is conducted sequentially, cell by cell. This approach ensures a decision-making process that reflects the cumulative impact of consumption choices on the overall error and latency.

Reward Function: The level consumptions, once determined, are implemented within the CKKS computation management environment. The latency and error (detailed in Section III-C) can be got from environment.

The reward function is pivotal in guiding the agent towards optimal CKKS levels. It is defined to quantify the trade-offs between latency and error. In the design of the reward function for our reinforcement learning model, we aim to incentivize the agent to minimize both latency and error while ensuring that the error level does not exceed the maximum acceptable threshold, $E_{\max}$. The function is asymmetrically weighted to prioritize latency reduction. To achieve this, we have formulated a reward function that takes into account the reduction in latency, the decrease in error, and adherence to the error upper limit.

The reward function is defined as follows:

$$R = \frac{L_0 - L}{L_0} + \alpha \times \log_{10} \left(\frac{E_{\max} - E}{E_0} \right) + \beta \times \frac{E}{E_0} \times I(E > E_{\max}) \quad (1)$$

Where:

- L_0 represents the initial latency before optimization.
- E_0 represents the initial error before optimization.
- L and E are the latency and error returned after the agent interacts with the environment, respectively.
- $E_{\max}$ is the maximum acceptable error.
- α and β are weighting parameters that adjust the emphasis on error reduction and compliance with the error threshold, respectively.
- $I(E > E_{\max})$ is an indicator function that takes a value of -1 when E exceeds $E_{\max}$, and 0 otherwise.

The first term of the reward function, $\frac{L_0 - L}{L_0}$, is given a higher weight to encourage the agent to reduce latency, which is a primary performance metric. The second term, $\alpha \times \log_{10} \left(\frac{E_{\max} - E}{E_0} \right)$, incentivizes the agent to decrease error, particularly as the error approaches the maximum acceptable level, by providing a higher reward for error reduction. The third term, $\beta \times \frac{E}{E_0} \times I(E > E_{\max})$, serves as a penalty to discourage the agent from actions that result in error exceeding

the threshold. This term assigns a negative reward (-1) when the error surpasses $E_{\max}$, thus ensuring that the agent avoids such actions. If the error remains within acceptable limits, this term contributes a value of 0 to the reward, imposing no penalty.

Through this reward function design, we guide the agent to balance the reduction of latency and error while strictly adhering to the error upper limit, thereby achieving effective system optimization. The asymmetric weighting clearly communicates the priority of latency reduction in the optimization process.

The training methodology employs the REINFORCE algorithm [23], renowned for its simplicity and proven effectiveness within the realm of policy gradient methods. The training regimen is structured around a series of iterative interactions between the agent and the environment. During each iteration, the agent actively explores the state space, making decisions based on the current state and continually refining its policy based on the rewards it receives.

C. CKKS Computation Management Environment

$$\begin{cases} \min_{k \leq 2S_f} \left(dp[i-1][k] + \min_{\substack{k \leq f \leq 2S_f \\ w \leq f, j}} E[i][f][w][j] \right) A = 0, B = 0 \\ \min_{k \leq j} \left(dp[i-1][k] + \min_{\substack{k \leq f \leq 2S_f \\ w \leq f, j}} E[i][f][w][j] \right) A = 0, B = 1 \\ \min_{k \leq 2S_f} \left(dp[i-1][k] + \min_{\substack{S_f \leq f \leq 2S_f \\ w \leq f, j}} E[i][f][w][j] \right) A = 1, B = 0 \\ \min_{k \leq j} \left(dp[i-1][k] + \min_{\substack{S_f \leq f \leq 2S_f \\ w \leq f, j}} E[i][f][w][j] \right) A = 1, B = 1 \end{cases} \quad (2)$$

During each episode, agent chooses the level consumption for each cell, the CKKS computation management returns the estimated latency and error under the level consumptions. The estimations are got from level management and error management respectively. The AceHomo environment first determines the cells' levels, and then specifies the I/O scales and waterlines for each cell to obtain the control flow graph (CFG) with the smallest inference error.

Level Management first directly utilizes an existing bootstrapping insertion scheme. With the cell as the smallest unit, the bootstrapping insertion schemes like Fhelipe [17] and DaCapo [16] can be easily adapted to AceHomo and finished within seconds. Upon to location of bootstrapping and level consumptions. The levels of all cells are also allocated. In Figure 3, as depicted, the plan level consumption of the fifth cell has been reduced compared to the original. This reduction in consumption has paved the way for the identification of new bootstrapping insertion points, which are crucial for defining the specific levels required for each cell in the system. We hypothesize that bootstrapping can refresh the ciphertext level to 7. For instance, considering the first ConvBN cell in

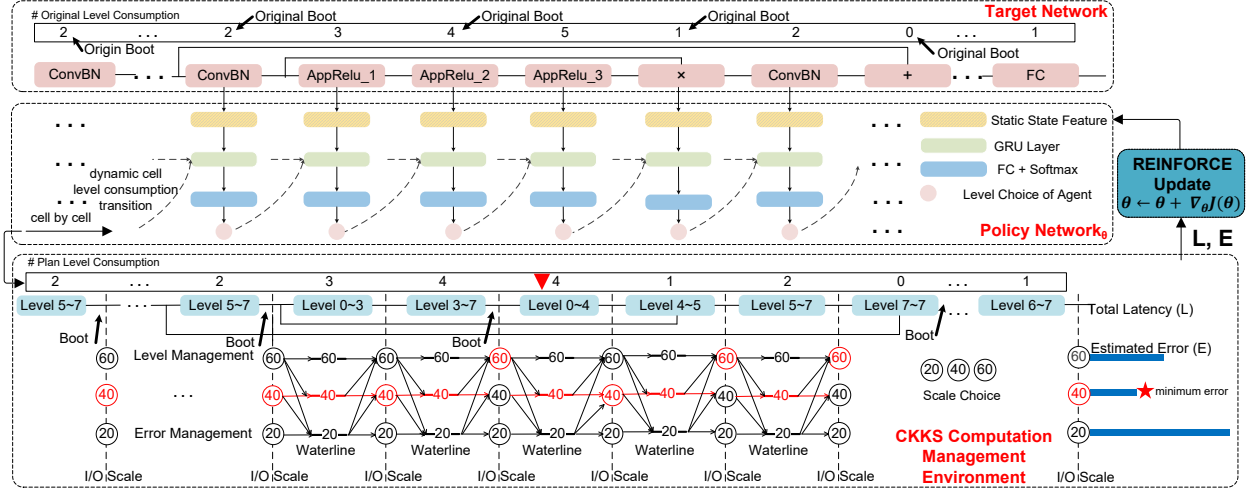


Fig. 3. AceHomo Reinforcement Learning Framework Overview. The policy agent determines the level consumption for each cell within the target neural network. The CKKS computation management environment applies level management and error management strategies, providing the agent with rewards based on estimated latency and error, thereby guiding the optimization process.

Figure 3, the bootstrapping insertion scheme indicates that a bootstrapping operation should be inserted immediately after it. With a level consumption of 2, the computational level for this cell extends from 5 to 7. Similarly, the computational levels for other cells are determined the same way.

Error management in AceHomo is designed to meticulously determine the I/O scale and waterline for each cell, aiming to minimize inference error. By employing a dynamic programming (DP) algorithm, we strategically minimize the inference error for a program comprising n cells. A key observation is that if the liveness variables at the completion of the i -th cell are all at the same scale j , the planning of I/O scales and waterlines for the first i cells become irrelevant to the subsequent cells. Furthermore, if all outputs of a cell are subjected to bootstrapping, they are reset to the scale S_f , rendering the I/O scale and waterline of that cell non-influential on future cells.

Leveraging this non-aftereffect property, we define the state $dp[i][j]$ as the inference error accumulated by the first i cells, given that the output scale of the i -th cell is j . Additionally, $E[i][f][w][j]$ quantifies the inference error contributed by the i -th cell, with an input scale of f , waterline w , and output scale j , as estimated by the error model in ELASM [9]. The variable A is assigned a value of 1 to indicate that the input to the i -th cell has undergone bootstrapping, and 0 otherwise. Similarly, variable B is employed to denote the presence of liveness variables from preceding cells upon the conclusion of the i -th cell's computations, with 1 signifying their existence and 0 their absence. The inference error for the i -th cell, $dp[i][j]$, is then calculated using the following formula:

Upon the completion of the DP process, the minimum value of $dp[n][i]$ for $i \leq 2S_f$ is identified as the program's minimal inference error. This value serves as a benchmark for determining the optimal I/O scales and waterlines for each cell. As depicted in Figure 3, following the level management's

execution of bootstrapping insertion, the error management module initiates the DP algorithm. The algorithm systematically evaluates the potential configurations to identify the plan that minimizes inference error, culminating in the selection of a strategy with an output scale of 2^{40} . In this optimal plan, the I/O scales and waterlines for each cell, pivotal for achieving the minimal error, are highlighted in red.

IV. EXPERIMENTS

A. Threat Model

We use the same threat model as previous privacy preserving works on FHE. The client sends encrypted data to an untrusted server. The server performs deep learning inference using FHE operations and returns the inference results to the client. The client decrypts the generated ciphertext to obtain the private result. The server holds only the client's public key and cannot decrypt any intermediate ciphertexts during the process. The client does not know any information about the server's processing other than the result.

B. Experimental Setup

Our experiments with AceHomo encompasses a thorough evaluation across varied neural network architectures: ResNet20 [24], ResNet32, ResNet44, and MobileNet [25]. These models are utilized to perform inference on the CIFAR-10 [26], thereby assessing efficacy within the AceHomo framework. Due to the substantial computational overhead inherent in homomorphic neural networks, prior research has predominantly focused on smaller datasets (e.g., CIFAR-10) and compact network architectures [10] [11] [13]. The extensive computation and storage requirements renders larger-scale datasets (e.g., ImageNet) currently infeasible for practical homomorphic evaluation. Our experimental setup aligns with established practices in the field. Within this RL framework, the discount factor γ is set to 0.99, emphasizing the significance of future rewards in the

TABLE II
SPEEDUP PERFORMANCE OF ACEHOMO WITH BASELINE MODELS

Model	Base Latency(s)	AceHomo Latency(s)	Speedup
Resnet20	9703	6602	1.47
Resnet32	16020	11666	1.37
Resnet44	22189	15580	1.42
MobileNet	25883	17590	1.47

TABLE III
COMPONENT-WISE LATENCY BREAKDOWN FOR RESNET20

Component	ConvBN	AppRelu	Boot	DS	AvgPool	FC	Total
Base	1292	393	7971	28	12	6	9703
AceHomo	817	432	5315	18	12	7	6602

agent's calculations. The α and β are adjusted based on the training of the reinforcement learning process and ultimately determined to be 0.05 and 0.1, respectively. Additionally, the maximum permissible error level $E_{\max}$ is adjusted to an order of magnitude higher than the base error level, specifically calibrated to 10 times the base.

Regarding the CKKS encryption parameters, we have established a polynomial modulus degree N of 2^{16} . The level for bootstrapping operations is designated as 15, while the level for computation is set to 7. Additionally, the bit length of the modulus for bootstrapping S_b is configured to 51, and for computation S_f is set to 46.

For establishing our baseline, we assume a uniform waterline S_f for all cells, specifically 2^{46} . Bootstrapping is carefully inserted manually, adhering to the strategic guidelines outlined in [11]. This approach aims to minimize the number of bootstrapping operations while prioritizing their placement after convolutional layers whenever possible. The experimental evaluations are executed on Intel Xeon E5-2650 CPU with 128GB of memory, and 4 Tesla V100 GPU.

C. AceHomo Speedup Performance Evaluation

The primary metric for evaluation is latency of each model with the AceHomo framework applied. The results are summarized in Table II, which delineates the inference latency for a single image. It presents a comparison of the baseline latency and the latency after applying AceHomo, along with the speedup achieved. To further understand the impact of AceHomo, we conducted a component-wise analysis of the ResNet20 model in Table III.

The results in Table II demonstrate that AceHomo consistently reduces the latency across all tested models, with speedups ranging from 1.37x to 1.47x. This indicates that AceHomo is effective in optimizing the performance of encrypted neural network inference. The component-level analysis in Table III further exposes a marked decrease in latency for the bootstrapping process, a pivotal aspect of homomorphic encryption operations. Additionally, there is a notable reduction in latency for the ConvBN, which collectively bolsters the model's overall performance. It is worth noting that while AceHomo generally leads to a decrease in latency, the

TABLE IV
ACCURACY COMPARISON OF ACEHOMO WITH BASELINES

Model	Base Accuracy	AceHomo Accuracy
Resnet20	91.5%	91.3%
Resnet32	92.4%	92.4%
Resnet44	92.7%	92.6%
MobileNet	91.4%	91.4%

AppRelu component exhibits a marginal increase in latency. A comprehensive exploration will be provided in the subsequent subsection.

D. CKKS Computation Management Analysis

Figure 4 delineates a comprehensive comparative analysis of ResNet20, highlighting the computational characteristics pre/post-AceHomo optimization. The initial state is depicted in the upper half, with the optimized configuration presented in the lower half. The optimization is instrumental in achieving a substantial decrease in level consumption for ConvBN and AppRelu cells, with a specific reduction in AppRelu levels from $\{4, 4, 5, 1\}$ to $\{3, 4, 4, 1\}$.

Figure 5 compares the waterline levels before and after AceHomo optimization for the first Conv layer and ReLU layer. AceHomo reduces the waterline of Conv layer to 2^{23} , thereby decreasing its level consumption by one. As analyzed in Figure 1, this Conv layer optimization introduces little accuracy loss during inference. However, optimizing Conv layers alone cannot improve Bootstrapping operation insertion. AceHomo strategically adjusts the waterline of ReLU layers without significantly increasing approximation errors, successfully reducing level consumption by one in each polynomial evaluation module. This optimization decreases Bootstrapping operations from 3 to 2, thereby substantially optimizing computational overhead.

AceHomo's optimization strategy is two-pronged: firstly, it minimizes the level consumption of ConvBN and AppRelu layers, thereby reducing the incidence of computationally intensive bootstrapping operations. Secondly, it strategically places bootstrapping operations post each convolutional layer. This strategic placement enables convolutional layers to function at the lowest possible level, effectively minimizing latency. Despite the repositioning of bootstrapping insertion points, which leads to ReLU layers being computed at elevated levels, the overall performance is enhanced rather than diminished.

E. Inference Accuracy

This subsection provides an evaluation of the inference accuracy of CKKS neural networks when optimized with AceHomo. The accuracy metrics, as outlined in Table IV, indicate that AceHomo has little impact on the performance of these models. Specifically, the accuracy of ResNet20 experienced a minor decline from 91.5% to 91.3%, while ResNet32 and MobileNet maintained its accuracy. These results suggest that AceHomo can preserve the inference accuracy

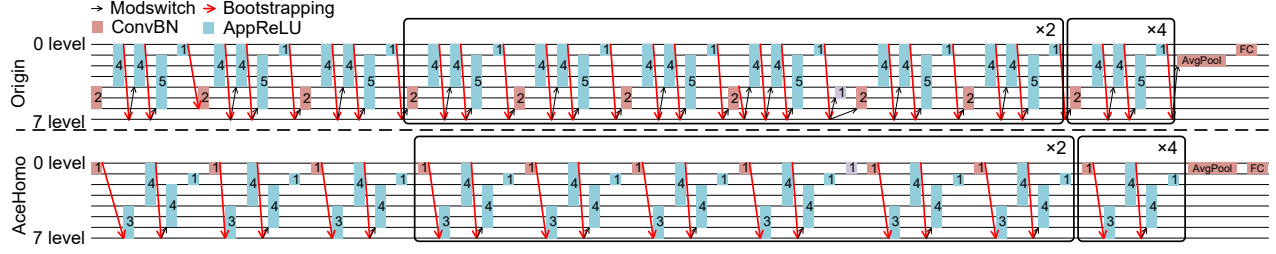


Fig. 4. Visualization of levels and level consumptions of ResNet20 with(out) AceHomo.

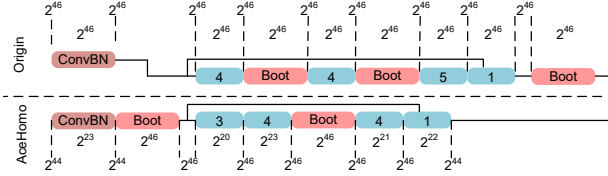


Fig. 5. Visualization of waterline of ResNet20 with(out) AceHomo.

of neural networks with minimal impact, as the changes in accuracy are negligible.

In summary, AceHomo's optimization approach not only effectively reduces the computational overhead but also manages error propagation through the strategic determination of operational scales. This meticulous optimization significantly reduces the latency, thereby enhancing computational efficiency without exerting a significant impact on the encrypted neural networks' accuracy.

V. CONCLUSION

This paper presents AceHomo, an innovative framework that employs reinforcement learning to refine CKKS encryption processes within neural networks. By adeptly navigating the interplay between CKKS computation and neural network robustness against error, AceHomo addresses the critical need for enhanced performance in privacy-preserving inference. It has proven its ability to significantly reduce latency without compromising network accuracy, offering a promising advancement in secure and efficient encrypted neural network inference.

REFERENCES

- [1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Symposium on the Theory of Computing*, 2009.
- [2] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," *Springer, Cham*, 2017.
- [3] J. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rms variant of approximate homomorphic encryption," *SAC (Conference)*, 2018.
- [4] J. W. Lee, H. C. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, and Y. S. Kim, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, 2021.
- [5] J. Zhang, J. Liu, L. He, X. Yang, Y. Wang, K. Chen, X. Hou, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," *IACR Cryptol. ePrint Arch.*, 2024.
- [6] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *IACR Cryptology ePrint Archive*, 2019.
- [7] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Cryptographers' Track at the RSA Conference*, 2020.
- [8] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musu-vathi, "Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *PLDL*, 2020.
- [9] Y. Lee, S. Cheon, D. Kim, D. Lee, and H. Kim, "Elasm: Error-latency-aware scale management for fully homomorphic encryption," in *USENIX Security*, 2023.
- [10] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *USENIX security*, 2018.
- [11] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," *Cryptology ePrint Archive*, Paper 2021/1688, 2021.
- [12] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. E. Lauter, M. Naehrig, and J. R. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016.
- [13] A. A. Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus," *IEEE Transactions on Emerging Topics in Computing*, 2021.
- [14] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, 2017.
- [15] Y. Lee, S. Heo, S. Cheon, S. Jeong, C. Kim, E. Kim, D. Lee, and H. Kim, "Hecate: Performance-aware scale optimization for homomorphic encryption compiler," in *CGO*, 2022.
- [16] S. Cheon, Y. Lee, D. Kim, J. M. Lee, S. Jung, T. Kim, D. Lee, and H. Kim, "Dacapo: Automatic bootstrapping management for efficient fully homomorphic encryption," in *USENIX Security Symposium*, 2024.
- [17] A. Krastev, N. Samardzic, S. Langowski, S. Devadas, and D. Sanchez, "A tensor compiler with automatic data packing for simple and efficient fully homomorphic encryption," *PACMPL*, 2024.
- [18] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017.
- [19] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018.
- [20] E. Lee, J.-W. Lee, J.-S. No, and Y.-S. Kim, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE TDSC*, 2021.
- [21] D. Kim, J. H. Park, J. Kim, S. Kim, and J. H. Ahn, "Hyphen: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, 2023.
- [22] J. H. Cheon, M. Kang, T. Kim, J. Jung, and Y. Yeo, "High-throughput deep convolutional neural networks on fully homomorphic encryption using channel-by-channel packing," *Cryptology ePrint Archive*, Paper 2023/632, 2023.
- [23] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 2004.
- [24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2015.
- [25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv, abs/1704.04861*, 2017.
- [26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Handbook of Systemic Autoimmune Diseases*, 2009.