Optimizing Encrypted Neural Networks: Model Design, Quantization and Fine-Tuning Using FHEW/TFHE

Yu-Te Ku[1,4], Feng-Hao Liu[2], Chih-Fan Hsu[1],
Ming-Ching Chang[3], Shih-Hao Hung[4,5], I-Ping Tu[4] ,Wei-Chao Chen[1]
[1] Inventec Corporation, Taipei City, 111059, Taiwan; [2] Washington State University, Pullman, WA, 99164, USA;
[3] State University of New York, University at Albany, Albany, NY 12222, USA;
[4] Data Science Degree Program, National Taiwan University and Academia Sinica, Taipei City, 115201, Taiwan;
[5] High Performance and Scientific Computing Center, National Taiwan University, Taipei City, 10617, Taiwan
d08946006@ntu.edu.tw; feng-hao.liu@wsu.edu; hsu.chih-fan@inventec.com; mchang2@albany.edu;
hungsh@csie.ntu.edu.tw; iping@stat.sinica.edu.tw; chen.wei-chao@inventec.com

Abstract

Third-generation Fully Homomorphic Encryption (FHE), particularly the FHEW/TFHE schemes, is recognized for its balanced security requirements, small parameters, and low memory usage, though the current methods in the scenarios of Deep Neural Network (DNN) inference still have high computational costs, limiting the practical applicability. This work demonstrates how to improve practicality of the third-generation technologies for DNN tasks while preserving its key advantages. Our work focuses on two main contributions. First, we developed a computational architecture called FHE-Neuron, which reconfigures the parameters and bootstrapping structure of traditional FHEW/TFHE Boolean operations. This architecture significantly reducing the cost of encrypted DNN inference by dynamically switching the precision of encrypted data during computation—using high precision for cost-effective linear operations and low precision for computationally expensive nonlinear operations. Second, we introduced an FHE-aware Quantization and Fine-tuning framework that optimizes model parameters to align with FHE-Neuron's constraints, ensuring high accuracy in encrypted inference. We validate our approach on various neural network models across several computing platforms. In our experiments, our method achieves one-image inference time on average 4.5 milliseconds for MNIST and 17 milliseconds for Fashion MNIST, achieving accuracy rates of 96.52% and 88.57% respectively. For the CIFAR-10 dataset, our system completes one image inference in 30 seconds with a 90.5% accuracy rate.

Keywords

Encrypted inference, neural networks, Fully Homomorphic Encryption, FHEW/TFHE, boostrapping, quantization, approximated computation, model fine-tuning.

1 Introduction

With the rapid rise of AI and deep learning, the deployment of Deep Neural Network (DNN) models through various platforms and services has advanced significantly [19, 23, 28, 42, 52, 55]. However, the widespread adoption of these technologies also raises concerns about data security, privacy, and potential model leaks. Fully Homomorphic Encryption (FHE) allows computations on encrypted data without the need for decryption, ensuring data security and privacy. Non-interactive FHE Encrypted DNN Inference [18, 40] enables users to securely outsource encrypted data for processing without exposing the data, addressing these challenges.

This work demonstrates that third-generation FHE technologies, derived or extended from FHEW [15] and TFHE [10], can achieve performance comparable to the highly optimized fourth-generation CKKS [8] scheme, despite using smaller parameter settings. This integration enables the combination the compact ciphertext sizes and low memory requirements of third-generation solutions with the enhanced throughput of fourth-generation systems. Currently, mainstream Fully Homomorphic Encryption (FHE) schemes fall into two categories: (1) those supporting Single Instruction Multiple Data (SIMD) arithmetic operations, such as second-generation schemes like BGV [5] and BFV [16], as well as fourth-generation schemes like CKKS, and (2) those capable of executing arbitrary Boolean logic, like third-generation schemes such as FHEW and TFHE. CKKS is widely used for encrypted inference due to its native support for approximate computations with large floating-point numbers and efficient SIMD usage. However, it requires larger ciphertext and key sizes because of its superpolynomial modulus-to-noise ratio, which is based on the Ring Learning With Errors (Ring-LWE) problem.

In contrast, third-generation FHE schemes such as FHEW and TFHE based on moderate assumptions, specifically the Ring-LWE problem with a polynomial modulus-to-noise ratio. This configuration enables smaller ciphertexts and key sizes, allowing for faster processing of a single ciphertext compared to fourth-generation schemes. However, the lack of practical SIMD implementations for FHEW/TFHE means that fourth-generation FHE often outperforms these earlier schemes in many DNN inference applications. Despite this, third-generation FHE schemes remain important for their potential to diversify DNN encrypted inference solutions. Recent theoretical work suggests the feasibility of SIMD in third-generation FHE schemes [32, 33], although no supporting libraries have yet been developed. This indicates that third-generation FHE could eventually achieve efficiencies comparable to those of fourth-generation SIMD. Despite the absence of SIMD functionality, third-generation FHE still faces several challenges in DNN inference

Proceedings on Privacy Enhancing Technologies 2025(4), 1075–1091

https://doi.org/10.56553/popets-2025-0172

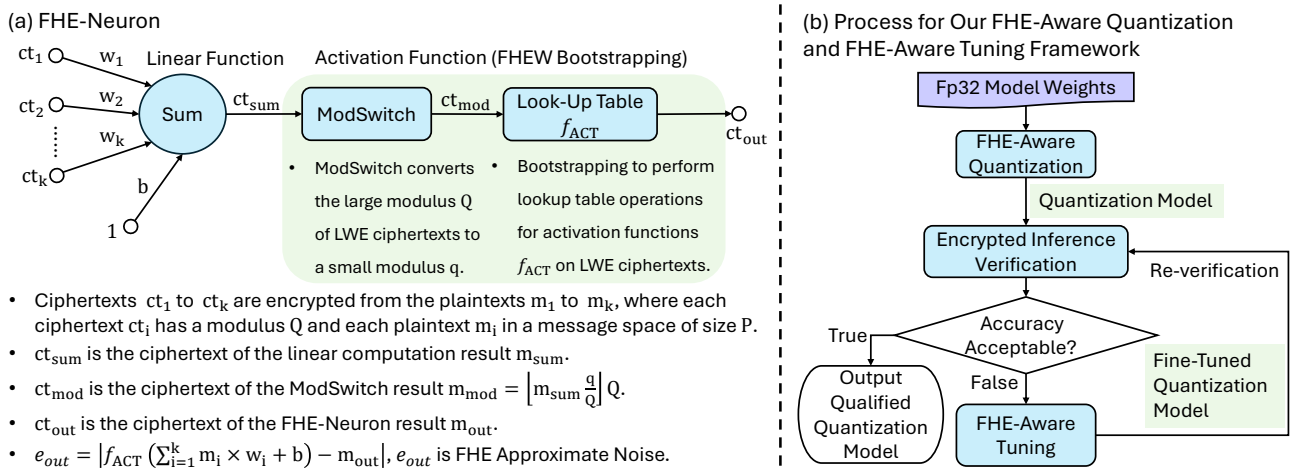

Figure 1: Overview of the proposed (a) **FHE-Neuron** leverages ModSwitch to switch the precision of ciphertexts, employing high precision for linear computations and low precision for activation functions, thereby enhancing efficiency. (§ 3) and (b) **FHE-Aware Quantization and Fine-tuning Framework** quantizes and fine-tunes pre-trained floating-point models to align with the specific usage conditions of FHE-Neuron. (§ 4).

applications. In the following sections, we explore new insights and potential solutions to address these challenges.

Challenges: The inherent limitations of weaker Ring-LWE assumption, i.e., small modulus-to-noise ratio, mainly come from the message space available for computations. For instance, the message space of FHEW/TFHE-based DNN inference scheme [26] is restricted to 6 bits, while CKKS-based [31] can support up to 60 bits. This limitation poses considerable challenges for designing efficient encrypted DNN computations.

Consequently, FHEW/TFHE-based design have proved effective only for simpler datasets like MNIST [4, 26], or they involved low-bit precision neural networks [17, 29], or they require converting computations to Boolean circuits [38]. However, the efficiency of these approaches is notably slower than those based on CKKS [3, 14, 22, 44, 45]. It remains unclear how to extend third-generation methods to efficiently handle larger datasets requiring more complex DNN models, such as CIFAR-10.

To address these challenges, we focus on the following **research question**: *Can we develop encrypted DNN inference based on the weaker Ring-LWE assumptions (e.g., polynomial modulus-to-noise ratio like the FHEW/TFHE schemes), while achieving comparable accuracy and computation time to highly optimized fourth-generation FHE schemes?*

This question is crucial in determining whether third-generation FHE schemes can offer a viable DNN inference solution that matches the performance of fourth-generation FHE schemes while utilizing smaller ciphertext sizes and requiring less memory.

1.1 Our Contributions

We address the aforementioned research question, and present an effective solution comprising two key components (see Figure 1). First, we design an efficient FHE-Neuron architecture to address the limitations of traditional Boolean FHEW/TFHE, which are unsuitable for DNN inference due to the high cost of message space. Second, we develop a model quantization and fine-tuning framework tailored to this architecture, enabling the conversion of general pre-trained models into ones compatible with our approach, thereby improving the robustness of encrypted inference. Finally, we validate our method on multiple datasets, including MNIST [30], Fashion-MNIST [53], and CIFAR-10 [27]. Experimental results demonstrate that within third-generation FHE technology, our approach significantly outperforms previous methods. We elaborate our contributions in details as follows.

Contribution 1: We design an efficient encrypted neural architecture, **FHE-Neuron**, based on third-generation FHE schemes, as shown in Figure 1(a). For further details, refer to § 3. The core idea is to optimize neuron computation efficiency by dynamically adjusting the precision of the ciphertexts. During the linear computation stage, we use large-modulus ciphertexts to support a larger message space. Before applying the activation function, we reduce the ciphertext modulus via FHEW ModSwitch operation to lower message precision. This adjustment is crucial because Bootstrapping, required for activation, becomes more computationally expensive with higher precision. Given DNN's interent tolerance for approximation errors, our approach maintains accuracy while significantly improving efficiency. Compared to traditional Boolean-type FHEW, which performs fully precise computations, our FHE-Neuron balances efficiency and accuracy through precision-switching strategies.

Contribution 2: We develop the **FHE-Aware Quantization and FHE-Aware Tuning Framework** to adapt plaintext-trained DNN models for efficient encrypted inference with FHE-Neurons. This framework overcomes key challenges of FHE-based computation, such as integer-only operations, limited ciphertext message space, and precision-related approximation errors. As shown in Figure 1(b), the **FHE-Aware Quantization** optimizes model precision based on FHE parameters to prevent overflow errors. The **FHE-Aware Tuning** module fine-tunes model weights using FHE parameters

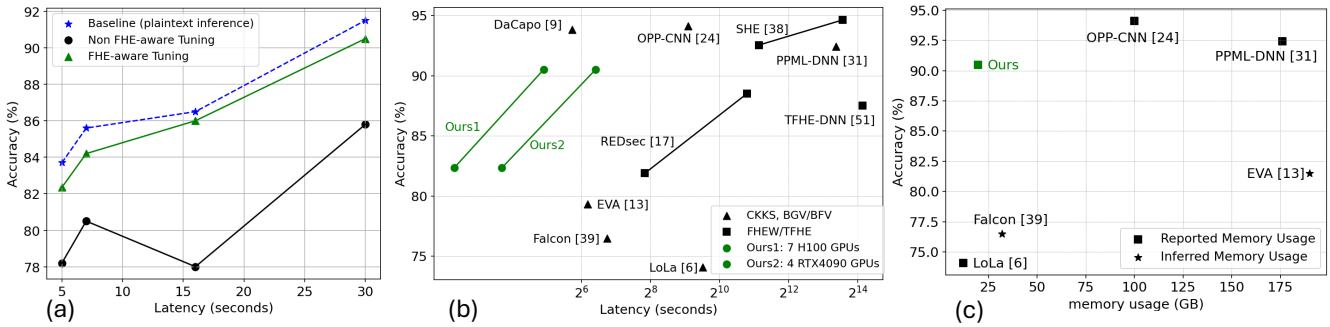

Figure 2: Performance and efficiency comparison with the State-of-The-Art (SoTA). Using 7 H100 GPUs or 4 RTX 4090 on the VGG9 model and CIFAR-10 dataset; see § 5 for details. (a) FHE-Aware Tuning significantly improves the accuracy of DNN encrypted inference. (b) Performance comparison of various encrypted DNN inference models on CIFAR-10, highlighting the trade-off between latency and accuracy. (c) Comparison of model memory usage and accuracy.

and the estimated noise distribution of FHE-Neuron, enhancing the reliability of encrypted computations. Ultimately, this framework ensures encrypted inference accuracy comparable to plaintext computation.

Contribution 3: We validate the effectiveness and efficiency of our approach through extensive experiments on the MNIST, Fashion MNIST, and CIFAR-10 datasets, using the OpenFHE [2] and GPU-accelerated FHEW/TFHE libraries [54]. The results demonstrate that after applying FHE-Aware Tuning, encrypted inference accuracy significantly improves, closely matching the plaintext inference accuracy of the original pre-trained models; see Figure 2(a). Compared to previous third-generation FHE (FHEW/TFHE)-based solutions, our method achieves the lowest latency in Figure 2(b). Additionally, while maintaining efficiency comparable to CKKS-based solutions, our approach offers lower memory consumption as in Figure 2(c). These findings confirm that third-generation FHE can enable practical and efficient encrypted DNN inference under weaker security assumptions, addressing primary research challenges.

Our work primarily focuses on improving efficiency of third-generation FHE in the application of DNN inferences. We notice that several techniques are also applicable to other FHE generations as we further elaborate in § 6.

1.2 Related Work

From 2018 to 2024, FHE has driven substantial progress in encrypted deep neural network (DNN) inference. The following studies specifically focus on third-generation FHE schemes. FHE-DiNN [4] demonstrated that FHEW/TFHE could be effectively applied to fully connected NNs on the MNIST dataset, achieving an inference time of 0.14 seconds per image with negligible accuracy degradation. FDFB [26] extended TFHE to support convolutional neural networks (CNNs) by introducing full-domain bootstrapping, representing an important step toward practical encrypted inference. SHE [38] extended TFHE to deep models like ResNet and LSTM, but inference was still expensive, with ResNet-18 taking about 3.3 hours. REDsec [17] improved runtime efficiency through discretized ciphertext representations, reducing CIFAR-10 inference time to approximately 2,000 seconds per image. EFHEPE [29] proposed a more efficient FHE framework tailored for privacy-enhanced neural inference in trustworthy AI applications.

Parallel research has explored alternative FHE schemes such as BGV [5]/BFV [16], and CKKS [8]. Falcon[39] and LoLa[6], both based on the BFV scheme, enhance encrypted inference efficiency through spectral-domain acceleration and optimized ciphertext representations. CKKS, due to its support for approximate arithmetic, has been widely adopted in privacy-preserving deep learning. PPML-DNN [31] applied CKKS to encrypted inference on CIFAR-10 but faced high computational costs—around three hours and 172 GB per image. EVA [13] streamlines FHE development by automating vectorization and operations. Hybrid-HE DNN framework [35] integrates the strengths of both the FHEW/TFHE and CKKS schemes to enhance the efficiency of encrypted DNN inference. OPP-CNN [24] mitigated this using constant-time convolutions, while DACAPO [9] further optimized performance through a compiler that inserts bootstrapping based on ciphertext scale, reducing inference time to under 60 seconds. CKKS has also been extended to encrypted graph convolutional networks (GCNs) [25]. CryptoGCN [46] enabled efficient inference on NTU-RGB+D [34]. Subsequent works such as Penguin [47] and LinGCN [43] improved scalability via parallel packing and linearized architectures.

2 Preliminary

In this section, we discuss preliminary knowledge necessary for our method. § 2.1, § 2.2, and § 2.3 cover the foundational aspects of third-generation FHE schemes, particularly FHEW/TFHE. § 2.4 and § 2.5 delve into the prerequisites for fine-tuning and quantizing DNN models.

2.1 LWE Symmetric Encryption

We begin by recalling the definition of the Learning with Errors (LWE) encryption scheme, which forms the foundation of many widely used Fully Homomorphic Encryption (FHE) schemes [15]. Let q denote the ciphertext modulus and P the message modulus. In the LWE scheme, a message $m \in \mathbb{Z}_P$ is encrypted with a secret key $\vec{\mathbf{s}} \in \mathbb{Z}_q^n$ in the form $\mathbf{ct}^{(P,q,n)} = (\vec{a}, \langle \vec{a}, \vec{\mathbf{s}} \rangle + \frac{q}{P} m + e) = (\vec{a}, b) \in$

$\mathbb{Z}_q^n \times \mathbb{Z}_q$, where $e \in \mathbb{Z}_q$ is a small error term, typically sampled from a Gaussian distribution. We use $\mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m\frac{q}{P}\rfloor\right)$ to denote the set of ciphertexts that encrypt m under secret key $\vec{\mathbf{s}}$, with respect to the moduli P, q.

Given an LWE ciphertext $(\vec{a}, b)$, the decrypted message m' can be computed as $m' = \left\lceil \frac{p}{q}\left(b - \langle \vec{a}, \vec{\mathbf{s}}\rangle\right)\right\rfloor$, where $\lceil\cdot\rfloor$ denotes a rounding function. Decryption remains correct as long as the norm of error term $|e|$ does not exceed $\frac{q}{2P}$. This constraint is crucial when designing homomorphic operations over LWE ciphertexts, as each operation increases the error, impacting the failure probability of decryption.

We next present several useful homomorphic operations to process the LWE ciphertexts.

- $\mathsf{ModSwitch}_{Q\to q}\left(\mathbf{ct}^{(P,Q,n)}\right) \to \mathbf{ct}_{\text{out}}^{(P,q,n)}$: Given a LWE ciphertext, $\mathbf{ct}^{(P,Q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,Q,n)}\left(\left\lceil m\frac{Q}{P}\right\rfloor\right)$, moduli Q, the operation produces an output ciphertext $\mathbf{ct}_{\text{out}}^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m\frac{q}{P}\rfloor\right)$, moduli q.
- $\mathsf{KeySwitch}_{N\to n}\left(\mathbf{ct}^{(P,q,N)}\right) \to \mathbf{ct}_{\text{out}}^{(P,q,n)}$: Given LWE ciphertext, $\mathbf{ct}^{(P,q,N)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,N)}\left(\lceil m\frac{q}{P}\rfloor\right)$, where $\vec{\mathbf{s}} \in \mathbb{Z}_q^{N+1}$ and $\vec{\mathbf{s}}' \in \mathbb{Z}_q^{n+1}$, the operation produces an output ciphertext $\mathbf{ct}_{\text{out}}^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}'}^{(P,q,n)}\left(\lceil m\frac{q}{P}\rfloor\right)$. KeySwitch converts a ciphertext that is encrypted under the original secret key $\vec{\mathbf{s}} \in \mathbb{Z}_q^{N+1}$ into another ciphertext that is encrypted under a new secret key $\vec{\mathbf{s}}' \in \mathbb{Z}_q^{n+1}$.
- $\mathbf{ct}_1^{(P,q,n)} + \mathbf{ct}_2^{(P,q,n)} \to \mathbf{ct}_{\text{out}}^{(P,q,n)}$: The operation of adding two LWE ciphertexts, $\mathbf{ct}_1^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m_1\frac{q}{P}\rfloor\right)$ and $\mathbf{ct}_2^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m_2\frac{q}{P}\rfloor\right)$, results in an output ciphertext $\mathbf{ct}_{\text{out}}^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil (m_1+m_2)\frac{q}{P}\rfloor\right)$.
- $\mathbf{ct}_1^{(P,q,n)} + \left\lceil m_2\frac{q}{P}\right\rfloor \to \mathbf{ct}_{\text{out}}^{(P,q,n)}$: The operation that adds one LWE ciphertext $\mathbf{ct}_1^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m_1\frac{q}{P}\rfloor\right)$ to a plaintext $m_2\frac{q}{P}$ produces an output ciphertext $\mathbf{ct}_{\text{out}}^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil (m_1+m_2)\frac{q}{P}\rfloor\right)$.
- $\mathbf{ct}_1^{(P,q,n)} \cdot m_2 \to \mathbf{ct}_{\text{out}}^{(P,q,n)}$: The operation of multiplying a LWE ciphertext $\mathbf{ct}_1^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m_1\frac{q}{P}\rfloor\right)$ by a scalar factor $m_2 \in \mathbb{Z}$ produces an output ciphertext $\mathbf{ct}_{\text{out}}^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil (m_1\cdot m_2)\frac{q}{P}\rfloor\right)$.

2.2 FHEW/TFHE Functional Bootstrapping

Functional Bootstrapping [12] is a key feature third-generation FHE schemes, such as FHEW [15] and TFHE [10]. This advanced capability enables direct execution of specific homomorphic lookup operations on ciphertexts during the bootstrapping phase, defined by an **extraction** function, without incurring additional computational overhead. Within the FHEW/TFHE frameworks, Functional Bootstrapping is particularly useful for performing operations in small integer domains, which are essential for Boolean circuits. We integrate this technique into our FHEW-style cryptosystems, following the bootstrapping method outlined in [36]. Our implementation is based on the OpenFHE library [2], incorporating additional performance optimizations proposed in [21] and [54].

Throughout this paper, we make extensive use of the Functional Bootstrapping procedure, denoted as $\mathsf{Boots}[f]$ for a given negacyclic extraction function f. The key properties of $\mathsf{Boots}[f]$, essential to our work, are summarized in the following definition.

Lemma 2.1 (Functional Bootstrapping). *For any LWE ciphertext $\mathbf{ct}^{(P,q,n)} \in \mathsf{LWE}_{\vec{\mathbf{s}}}^{(P,q,n)}\left(\lceil m\frac{q}{P}\rfloor\right)$ and a negacyclic extraction function $f : \mathbb{Z}_q \to \mathbb{Z}_Q$, such that $f\left(\mathbf{x}+\frac{q}{2}\right) = -f(\mathbf{x}) \pmod Q$, the functional bootstrapping procedure $\mathsf{Boots}[f](\mathbf{ct}^{(P,q,n)})$ outputs a new ciphertext $\mathbf{ct}^{(P,Q,N)} \in \mathsf{LWE}_{\vec{\mathbf{s}}'}^{(P,Q,N)}\left(\left\lceil m'\frac{Q}{P}\right\rfloor\right)$, $m' = \left\lceil f(m)\frac{Q}{P}\right\rfloor + e \pmod Q$, with $P < q < Q$. This process transforms a ciphertext encrypted under the original secret key $\vec{\mathbf{s}} \in \mathbb{Z}_q^{n+1}$ into a ciphertext encrypted under a new secret key $\vec{\mathbf{s}}' \in \mathbb{Z}_Q^{N+1}$. The noise term e satisfies $|e| < \beta$, where β is a noise bound depending only on the operations performed by* Boots *and not on the input ciphertext $\mathbf{ct}^{(P,q,n)}$.*

2.3 FHE Parameters of Our FHE-Neuron

We adopt the FHEW/TFHE Functional Bootstrapping notation and parameter settings from [36] for our FHE-Neuron, as summarized in Table 1. Notably, we align the modulus of the initial LWE ciphertext, Q_{in}, with the LWE/RLWE modulus used for key switching, Q_{ks}.

The key input parameter for the FHE-Neuron is Q_{in}. Typically, $\log Q_{\text{in}}$ must exceed $\log P + \log e_{\text{in}}$, where $\log P$ represents the bit precision of the plaintext input, and e_{in} denotes the error in the input ciphertext $\mathrm{ct}_{\text{in}}^{(P,Q_{\text{in}},n)}$ as defined in Theorem 3.1. We set $\log Q_{\text{in}} = 35$ and $\log P = 16$.

Once Q_{in} is set, we select a modulus Q larger than Q_{in} to enable efficient NTT/FFT operations during bootstrapping. The ring dimension N is chosen based on the desired security level, following the LWE estimator recommendation in [1]. To achieve 128-bit security against classical attacks, we set $N = 2048$, with a maximum $\log Q$ of 54 bits. Given N, we set $q = 2N$ to optimize performance.

Along with selecting Q, we must determine B_g, the gadget base used for decomposing Q. Prior studies [36, 41] recommend setting B_g to the smallest power of two greater than $Q^{1/2}$, corresponding to a decomposition depth of $d_g = 2$. This choice is crucial, as B_g directly influences the growth of bootstrapping noise. To optimize runtime performance, we set $B_g = \lceil Q^{1/2}\rceil$.

Selecting an optimal B_g is essential for controlling the output error of the FHE-Neuron, ensuring minimal impact on DNN inference accuracy while maintaining computational efficiency. Our strategy balances precision and performance, avoiding unnecessary computational overhead. For example, experimental results show that setting $B_g = \lceil Q^{1/3}\rceil$ (with $d_g = 3$) increases computation time by 1.5× compared to $B_g = \lceil Q^{1/2}\rceil$ (with $d_g = 2$), while reducing noise standard deviation to approximately two-thirds of its original value.

2.4 Post-Training Quantization

Third-generation FHE schemes support only integer computations, requiring model weights to be converted from floating-points into integers. Post-Training Quantization (PTQ) [20] achieves this using a uniform quantization function $Q(w)$ defined as:

$$Q(w) = \text{sign}(w)\cdot\Delta\cdot\min\left(\left\lfloor\frac{|w|}{\Delta}+0.5\right\rfloor, \frac{M-1}{2}\right). \quad (1)$$

Table 1: Main FHE Parameters Related to FHEW/TFHE Functional Bootstrapping

Parameter	Description
P	Size of the plaintext message space for the initial LWE ciphertext.
q	Input the modulus of the LWE ciphertext for Bootstrapping.
n	LWE dimension of the initial LWE ciphertext.
Q	RLWE modulus of the Bootstrapping Key
Q_{in}	LWE Modulus of the initial LWE ciphertext.
Q_{ks}	LWE/RLWE modulus used for key switching.
N	RLWE dimension of the Bootstrapping Key.
B_g	Gadget base for digit decomposition in each accumulator update, which breaks integers mod Q into d_g digits.
B_{ks}	Gadget base for key switching, which breaks integers mod Q into d_{ks} digits.

Here, w represents the floating-point model weights, and $\text{sign}(w)$ determines its polarity. The quantization step Δ defines the spacing between quantization levels, while M denotes the total quantization levels. For example, with signed 8-bit integer for quantization, $M = 255$. This approach ensures a symmetric, uniform distribution of weights while minimizing quantization errors, making it well-suited for encrypted deep learning computations.

2.5 Pseudo-Noise Tuning

The primary goal of Pseudo-Noise Tuning (PNT) is to mitigate the loss of inference accuracy that often follows model quantization due to reduced precision. This technique introduces Pseudo-Quantization Noise (PQN) during the fine-tuning phase, enabling the model to adapt to and anticipate precision losses caused by quantization, ultimately improving inference accuracy.

A key component of this process is the **noise proxy function**, which adjusts pseudo-noise based on quantized network parameters and predefined noise distributions. During forward propagation, pseudo-noise is strategically added after each ReLU layer via this proxy function. The remaining fine-tuning steps follow standard procedures, as detailed in [7, 50]. This approach optimizes model weights to compensate for noise effects introduced by quantized parameters and pseudo-noise distributions. A more detailed discussion of pseudo-noise tuning can be found in Appendix B.

Our method extends PNT by estimating pseudo-noise distributions based on FHE theory. The impact of this approach and its benefits are further explored in § 4.2.

3 The FHE-Neuron

In this section, we present detailed design of our FHE-Neuron to address the challenges outlined in the Introduction. § 3.1 provides an overview of FHE-Neuron architecture and compare it with the traditional Boolean FHEW/TFHE scheme. § 3.2 provides detailed analysis of the FHE-Neuron.

3.1 Architecture of FHE-Neuron

Our FHE-Neuron process is outlined in Algorithm 1. The algorithm takes as input a $1 \times k$ ciphertext matrix $\vec{\mathbf{ct}}$, plaintext neuron weights $\vec{w} \in \mathbb{Z}^k$, a bias term $b \in \mathbb{Z}$, and an activation function f_{ACT}. The output ciphertext $\mathbf{ct}_{\text{out}}$ represents the plaintext m_{out}, computed by applying a linear operation to the plaintext embedded in $\vec{\mathbf{ct}}$, using weights $\vec{w}$ and bias b, followed by activation f_{ACT} and scaling by δ.

Algorithm 1 FHE-Neuron

Parameters:
- $Q_{in} \in \mathbb{Z}$: input LWE modulus.
- $P \in \mathbb{Z}$: message space modulus of input LWE.
- $n \in \mathbb{Z}$: dimension of input LWE.
- $q \in \mathbb{Z}$: input modulus for bootstrapping.
- $Q \in \mathbb{Z}$: RLWE modulus of the Bootstrapping Key.
- $N \in \mathbb{Z}$: RLWE dimension of the Bootstrapping Key.
- $\delta \in \mathbb{R}$: scale factor.

Input:
- $\vec{\mathbf{ct}} = [\mathbf{ct}_1^{(P,Q_{in},n)}, \ldots, \mathbf{ct}_k^{(P,Q_{in},n)}]$, input FHE LWE ciphertexts.
- $\vec{w} \in \mathbb{Z}^k$, model weights.
- $b \in \mathbb{Z}$, model bias.
- $f_{\text{ACT}} : \mathbb{Z}_P \to \mathbb{Z}_P$, an activation function.

procedure FHE-Neuron($\vec{\mathbf{ct}}, \vec{w}, b, f_{\text{ACT}}$)
▷ Convert activation function $f_{\text{ACT}} : \mathbb{Z}_P \to \mathbb{Z}_P$ into a negacyclic activation function $f_{\text{Bt}} : \mathbb{Z}_q \to \mathbb{Z}_Q$.
  function $f_{\text{BT}}(x)$
    if $\frac{-q}{4} \le x < \frac{q}{4}$ **then**
      return $\left\lfloor \delta \frac{Q}{q} \cdot f_{\text{ACT}}(\left\lfloor x \frac{2P}{q} \right\rfloor) \frac{q}{2P} \right\rfloor \mod Q$
    else
      return $-f_{\text{Bt}}(\lfloor x - \frac{q}{2} \rfloor)$
    end if
  end function

▷ Start FHE Linear Function
  $\mathbf{ct}_{\text{in}}^{(P,Q_{in},n)} \leftarrow \sum_{i=1}^{k} \mathbf{ct}_i^{(P,Q_{\text{in}},n)} \cdot \mathsf{w}_i + \left\lfloor \frac{b \cdot Q_{\text{in}}}{p} \right\rfloor$
▷ Start FHE Activation Function (FHE-ACT)
  $\mathbf{ct}_{\text{M1}}^{(q,q,n)} \leftarrow \mathsf{ModSwitch}_{Q_{\text{in}} \to q}(\mathbf{ct}_{\text{in}}^{(P,Q_{\text{in}},n)})$
  $\mathbf{ct}_{\text{Bt}}^{(P,Q,N)} \leftarrow \mathsf{Boots}[f_{\text{Bt}}](\mathbf{ct}_{\text{M1}}^{(q,q,n)})$
  $\mathbf{ct}_{\text{M2}}^{(P,Q_{in},N)} \leftarrow \mathsf{ModSwitch}_{Q \to Q_{in}}(\mathbf{ct}_{\text{Bt}}^{(P,Q,N)})$
  $\mathbf{ct}_{\text{out}}^{(P,Q_{in},n)} \leftarrow \mathsf{KeySwitch}_{n \to n}(\mathbf{ct}_{\text{M2}}^{(P,Q_{in},n)})$
end procedure
Output: $\mathbf{ct}_{\text{out}}^{(P,Q_{in},n)}$, an LWE ciphertext.

As illustrated in Figure 3(a), our FHE-Neuron architecture utilizes ModSwitch technology to switch ciphertext precision, thus optimizing computational efficiency. For linear computations, we employ ciphertexts with larger moduli and higher precision, while for nonlinear computations, we opt for smaller moduli and lower precision. This strategy is implemented because linear computations are more efficient than nonlinear operations executed via FHEW Bootstrapping in the LWE ciphertext environment. The nonlinear computation is referred to as **FHE Activation Function (FHE-ACT)**.

To minimize the costs of nonlinear computations, we implemented a simplified form of bootstrapping that supports only lookup table functionality for negacyclic functions. While more advanced bootstrapping methods can support lookup tables for any function, their implementation would significantly increase costs. We define activation functions like ReLU as negacyclic functions as in Figure 5. This negacyclic ReLU function effectively operates within $\pm\frac{q}{4}$ of

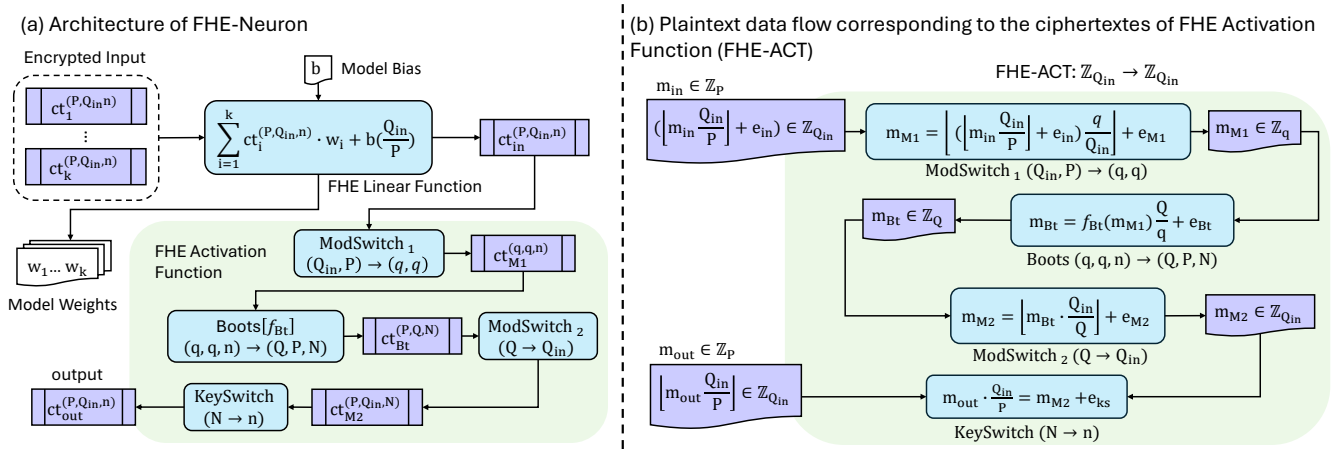

Figure 3: (a) Architecture of FHE-Neuron and (b) Noise analysis of FHE Activation Function (FHE Parameters: $q = 4096, n = 1305, N = 2048, P = 2^{16}, Q_{in} = 2^{35}, Q = 2^{54}$).

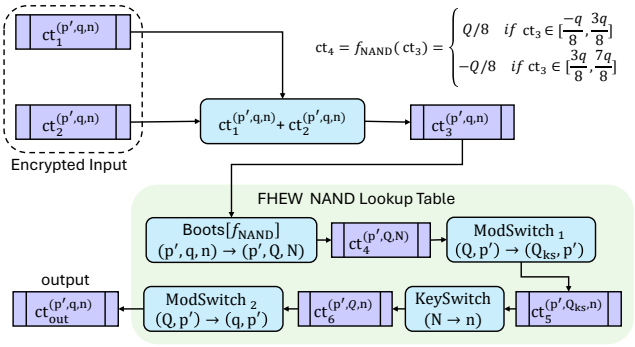

Figure 4: Traditional Boolean FHEW: A Case Study with NAND. (FHE Parameters: $q = 1024, n = 512, N = 1024, p' = 2^3, Q_{ks} = 2^{14}, Q = 2^{27}$).

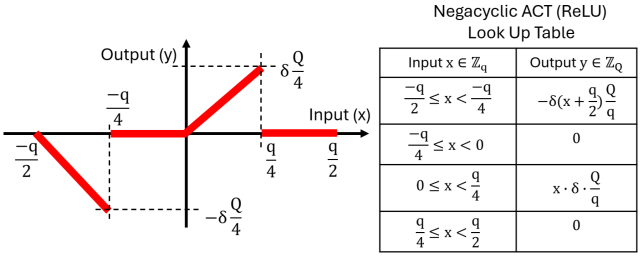

Figure 5: Negacyclic Activation Functions for Bootstrapping (using ReLU as an Example) $f_{Bt} : \mathbb{Z}_q \to \mathbb{Z}_Q$.

the ciphertext message space, covering only half of it. Exceeding this range during processing may lead to inference errors. In line 4 of Algorithm 1, the activation function f_{ACT} is transformed into the negacyclic function f_{Bt} for Bootstrapping. Furthermore, to handle message scaling in quantized DNNs, we multiply the ciphertext by a scaling factor δ during the Bootstrapping operation, adjusting the ciphertext information, where $0 < \delta \leq 1$.

Our FHE-Neuron employs mixed-precision computation to enhance performance, building on extensions of the traditional Boolean FHEW architecture (see Figure 4). The following paragraph outlines the key insights behind our approach.

Key Insights of Our New FHE-Neuron. We present the critical ideas of our extensions to FHE-Neuron, emphasizing the key differences as follow.

(1) FHE Parameters Settings: The LWE ciphertext moduli differ between the two architectures. Traditional Boolean FHEW prioritizes computational efficiency with a smaller modulus (*e.g.*, $q = 2^{10}$), limiting computations to low-bit values (*e.g.*, 3-bit). In contrast, FHE-Neuron supports higher precision (*e.g.*, 16-bit) for deep neural networks, requiring a larger LWE ciphertext modulus (*e.g.*, $Q\_in = 2^{35}$) to store more complex information for linear computations.

(2) Bootstrapping Mechanism: Both approaches use lookup tables during Bootstrapping after LWE-based linear computations. The efficiency and precision of Bootstrapping depend on the RLWE key dimension N. To balance security and performance, we set $N = 2048$ and limit the Bootstrapping input modulus to $q \leq 2N$. Increasing q would require a larger N, significantly raising computational costs. Consequently, FHE-Neuron optimizes performance by using a larger modulus (*e.g.*, 2^{35}) for linear computations and reducing it (*e.g.*, to 4,096) before Bootstrapping, thus minimizing computational costs. This reduction results in information loss of $\log(Q_{in}/q)$ bits from the ciphertext, but still supports a lookup table of $\log(q)$ bits. In contrast, traditional Boolean FHEW prevents information loss by aligning the ciphertext modulus used in linear computations with that used during bootstrapping. However, this restriction limits the computations to only Boolean values or small integers.

(3) LWE Ciphertext Scaling: After Bootstrapping, FHE-Neuron scales ciphertext messages by multiplying them with $\delta \in \mathbb{R}$, which is important for DNN operations. Traditional Boolean FHEW relies on costly Boolean circuits for numerical scaling, whereas our

approach achieves similar functionality without added cost. However, this introduces noise due to approximation errors, making it essential to analyze their impact on accuracy.

These distinctions highlight the advantages of FHE-Neuron in precision, efficiency, and deep learning adaptability. In § 3.2, we analyze FHE Activation Function, the primary source of approximation errors in our architecture.

3.2 Analysis on FHE-Neuron

In this section, we analyze the approximation errors encountered in Algorithm 1 (FHE-Neuron), with a particular emphasis on the FHE Activation Function (FHE-ACT) process spanning lines 12 to 15. The errors stemming from the Linear Function on line 11 are negligible. The primary sources of approximation errors are the changes in ciphertext message precision occurring between linear and nonlinear computations via ModSwitch, and the noise introduced by the scaling factor δ used in $\mathsf{Boots}[f_{\mathrm{Bt}}]$.

Figure 3 (b) shows the plaintext data flow corresponding to the ciphertexts of FHE-ACT, highlighting how noise is introduced through two ModSwitch operations, Bootstrapping, and KeySwitch. In particular, $\mathsf{ModSwitch}_1$ reduce modulus of the ciphertext from $Q = 2^{35}$ to $q = 2^{12}$, thus thus preserving only the top 12 bits of information in the ciphertext and discarding the lower bits. This effect is especially significant when the ciphertext message space $P = 2^{16}$ exceeds q, as the impact of $\mathsf{ModSwitch}_1$ on the message is more pronounced than in cases where the message space is smaller than q. Furthermore, Bootstrapping introduces noise during its lookup operations, where the impact of the noise is inversely proportional to Bootstrapping scaling factor δ; the smaller the scaling factor, the greater the noise impact. Therefore, meticulous control of the output range and noise ratio of the ciphertext through the scaling factor is essential for maintaining accurate inference results.

Theorem 3.1 (Noise Analysis of FHE Activation Function). *Let Q_{in}, P, q, Q denote parameters specified in Algorithm 1, arranged such that $Q > Q_{in} > P > q$. Then for ciphertext $\mathbf{ct}_{in}^{(P,Q_{in},n)} \in \mathsf{LWE}_{\vec{s}}^{(P,Q_{in},n)}\left(\left\lfloor m_{in}\frac{Q_{in}}{P}\right\rfloor\right)$, where $m_{in} \in \mathbb{Z}_P$ is the plaintext message and activation function $f_{ACT} : \mathbb{Z}_P \to \mathbb{Z}_P$ and nega- cyclic activation function $f_{Bt} : \mathbb{Z}_q \to \mathbb{Z}_Q$, the output ciphertext satisfies $\mathbf{ct}_{out}^{(P,Q_{in},n)} \in \mathsf{LWE}_{\vec{s}}^{(P,Q_{in},n)}\left(\left\lfloor m_{out}\frac{Q_{in}}{P}\right\rfloor\right)$, where m_{out} is the plaintext message, satisfying*

$$m_{out}\cdot\frac{Q_{in}}{P} = \left\lfloor f_{Bt}\left(\left\lfloor\left(\left\lfloor\frac{m_{in}\cdot Q_{in}}{P}\right\rfloor + e_{in}\right)\frac{q}{Q_{in}}\right\rfloor + e_{M1}\right)\frac{Q_{in}}{Q}\right\rfloor + e_{Bt}\frac{Q_{in}}{Q} + e_{M2} + e_{ks} \in \mathbb{Z}_{Q_{in}},$$

where $e_{in} \in \mathbb{Z}_{Q_{in}}$, $e_{M1} \in \mathbb{Z}_q$, $e_{Bt} \in \mathbb{Z}_Q$, $e_{M2} \in \mathbb{Z}_{Q_{in}}$, $e_{ks} \in \mathbb{Z}_{Q_{in}}$, and $e_{out} \in \mathbb{Z}_{Q_{in}}$ are defined as shown in Table 2. These terms are the commonly used noise components in FHEW/TFHE schemes.

Define $e_{out} := |f_{ACT}(m_{in}) - m_{out}|\,\frac{Q_{in}}{P}$ and then we have:

$$e_{out} = e'_{Bt} + e_{Bt}\frac{P}{Q} + e_{M2}\frac{P}{Q_{in}} + e_{ks}\frac{P}{Q_{in}} \tag{2}$$

$$= e_{M1}\frac{P}{q}\delta + e_{Bt}\frac{P}{Q} + e_{M2}\frac{P}{Q_{in}} + e_{ks}\frac{P}{Q_{in}} \in \mathbb{Z}_P, \tag{3}$$

where the error term:

$$e'_{Bt} = \left| f_{ACT}(m_{in})\delta - \left\lfloor f_{Bt}\left(\left\lfloor\left(\left\lfloor\frac{m_{in}\cdot Q_{in}}{P}\right\rfloor + e_{in}\right)\frac{q}{Q_{in}}\right\rfloor + e_{M1}\right)\frac{P}{Q}\right\rfloor\right| \approx e_{M1}\frac{P}{q}\delta, \text{ when } P > q \text{ and } f_{ACT}, f_{Bt} \text{ are smooth.}$$

Proof. We prove the theorem by tracing the value encrypted by the input ciphertexts $\mathbf{ct}_{in}^{(P,Q_{in},n)} \in \mathsf{LWE}_{\vec{s}}^{(P,Q_{in},n)}\left(\left\lfloor m_{in}\frac{Q_{in}}{P}\right\rfloor\right)$ as depicted in Algorithm 1 and Figure 3(b) . Line 4 executes the conversion of the activation function lookup table $f_{\mathrm{ACT}} : \mathbb{Z}_P \to \mathbb{Z}_P$ into a negacyclic activation function lookup table for bootstrapping $f_{\mathrm{Bt}} : \mathbb{Z}_q \to \mathbb{Z}_Q$. Line 11 retrieves $\mathbf{ct}_{in}^{(P,Q_{in},n)}$ through the FHE Linear Function. The noise e_{in} originates within the Learning with Errors (LWE) scheme of $\mathbf{ct}_{in}^{(P,Q_{in},n)}$, and belongs to the ring of integers modulo Q_{in}. Line 12 introduces noise $e_{\mathbf{M1}}$ during the initial Modulus Switch. Line 13 executes the bootstrapping procedure $\mathsf{Boots}[f_{\mathrm{Bt}}]$ and incorporates noise e_{Bt}. Line 14 introduces additional noise $e_{\mathbf{M2}}$ during the second Modulus Switch. Line 15 conducts the Key Switch and adds noise e_{ks}. Ultimately, the error between the input and output of the FHE-ACT, denoted as e_{out}, is given by Eq. (3).

□

Table 2: Noise Components and Their Descriptions

$e_{\mathrm{in}} \in \mathbb{Z}_{Q_{\mathrm{in}}}$	The original noise of LWE
$e_{M1} \in \mathbb{Z}_q$	The noise of $\mathsf{ModSwitch}_1$
$e_{\mathrm{Bt}} \in \mathbb{Z}_Q$	The noise of Boots
$e_{M2} \in \mathbb{Z}_{Q_{\mathrm{in}}}$	The noise of $\mathsf{ModSwitch}_2$
$e_{\mathrm{ks}} \in \mathbb{Z}_{Q_{\mathrm{in}}}$	The noise of KeySwitch
$e_{\mathrm{out}} \in \mathbb{Z}_{Q_{\mathrm{in}}}$	The noise of FHE-ACT

Empirical Estimation of FHE-ACT Noise: We now introduce our method to estimate the noise e_{out} in FHE-ACT. Based on the research documented in [36] and [54], we use independent Gaussian models to estimate the noise associated with Bootstrapping as implemented through the Number Theoretic Transform (NTT) and the Fast Fourier Transform (FFT).

NTT-based Implementation: According to Section 6.5 of [36], which discusses the Bootstrapping analysis based on NTT implementation, the error terms e_{M1}, e_{Bt}, e_{M2}, and e_{ks} are approximated as Gaussian distributions, each characterized by a specific standard deviation that describes their noise distribution. Their respective variances are $\sigma_{M1}^2 = \frac{(\|s_n\|^2+1)}{3}$, $\sigma_{M2}^2 = \frac{(\|s_N\|^2+1)}{3}$, $\sigma_{Bt}^2 = \frac{4d_g nN(B_g^2)}{6\sigma_{BK}^2}$, and $\sigma_{ks}^2 = \sigma_{BK}Nd_{ks}$, for uniform ternary secret keys s_N with dimensions N and s_n with dimensions n, $\|s_N\| \le \sqrt{N/2}$ and $\|s_n\| \le \sqrt{n/2}$, as estimated in [41]. From this information, we can approximate the total error $e_{\mathrm{out}} = \mathcal{N}(0, \sigma_{\mathrm{out}}^2)$, which follows a normal distribution with a mean of zero and a variance

$$\sigma_{\mathrm{out}}^2 \approx \sigma_{M1}^2\left(\frac{P}{q}\delta\right)^2 + \sigma_{Bt}^2\left(\frac{P}{Q}\right)^2 + \sigma_{M2}^2(\frac{P}{Q_{in}})^2 + \sigma_{ks}^2(\frac{P}{Q_{in}})^2 \in \mathbb{Z}_P. \tag{4}$$

FFT-based Implementation: Compared to NTT, bootstrapping implemented with FFT requires greater attention to noise issues due

to floating point precision losses in polynomial multiplication. We refer to the detailed analysis of FFT-based bootstrapping provided in [54]. The error introduced by FFT-based bootstrapping, e_{Bt-FFT}, can be approximated by a normal distribution with a standard deviation of σ_{Bt-FFT}, which incorporates the FFT bias δ_{FFT}, as expressed below: $e_{Bt-FFT} \sim \mathcal{N}(0, \sigma^2_{Bt-FFT})$ where $\sigma^2_{Bt-FFT} = (\sigma_{Bt} + \delta_{\text{FFT}})^2$.

The FFT bias, δ_{FFT}, originates from ciphertext decryption and is given by: $\delta_{\text{FFT}} = |(1 - \|s_n\|)\varepsilon_{\text{FFT}}|$, with $\|s_n\| \leq \sqrt{\frac{n}{2}}$ as described in [41]. The term ε_{FFT} represents the precision loss for each element in the LWE ciphertext and is defined as: $\varepsilon_{\text{FFT}} = \sqrt{n}\left\lfloor \sqrt{2u \cdot d_g} \times \frac{B_g \cdot Q \cdot \sqrt{N}}{4} \cdot e \right\rfloor$, where e is the relative error that depends on N and the specific FFT library used. For error analysis in the context of FFT-based bootstrapping applied to the FHE-ACT, simply replace σ_{Bt} in the formula with σ_{Bt-FFT}.

A Case Study: We provide an example to illustrate our estimation of the output error, e_{out}, for the NTT based FHE-ACT as defined in Theorem 3.1. We assume the following parameters for FHE: $q = 4,096$, $Q_{in} = 2^{35}$, $Q = 2^{54}$, $N = 2,048$, and $n = 1,305$. These parameters also include $B_g = \lceil Q^{1/2} \rceil$ (where $d_g = 2$), $d_{ks} = 7$, $\sigma_{ks} = 3.19$, and a smooth activation function f_{ACT} such as ReLU. We calculate $\sigma_{M1} = 123,808,958$, $\sigma_{M2} = 18.4842$, $\sigma_{ks} = 381.9483$, and $\sigma_{Bt} = 808,239,676,731$.

When using the FHE-ACT with the activation function f_{ACT} set to ReLU, and an input message $m_{in} = 20,000$ along with a scaling factor $\delta = 1$, the output message's standard deviation is approximately $\sigma_{\text{out}} \approx 236.1651$. This results in an output message of $m_{out} = 20,000 + \mathcal{N}(0, 236.1651^2)$. When the scaling factor is reduced to $\delta = 0.002$, the output standard deviation drops to approximately $\sigma_{\text{out}} \approx 2.978$, and the output message becomes $m_{out} = 40 + \mathcal{N}(0, 2.978^2)$.

After adjusting B_g to $\lceil Q^{1/3} \rceil$ (where $d_g = 3$), σ_{Bt} is updated to 1,933,373,826. This modification leads to a slight adjustment in the output standard deviation to $\sigma_{\text{out}} \approx 236.1468$ when $\delta = 1$, updating the output message to $m_{out} = 20,000 + \mathcal{N}(0, 236.1468^2)$. With $\delta = 0.002$, the output standard deviation further decreases to about $\sigma_{\text{out}} \approx 0.4723$, resulting in an output message of $m_{out} = 40 + \mathcal{N}(0, 0.4723^2)$. In all scenarios, σ_{out} follows a normal distribution $\mathcal{N}(0, \sigma^2_{\text{out}})$.

In Eq. (4), the terms σ^2_{M1} and σ^2_{Bt} are multiplied by the weighting factors $\left(\frac{p}{q}\delta\right)^2$ and $\left(\frac{p}{Q}\right)^2$ respectively. When $\delta = 1$, the primary source of noise in σ^2_{out} stems from σ^2_{M1}, thus reducing B_g does not significantly affect σ^2_{out}. However, when δ is reduced to 0.002, the predominant noise contributor becomes σ^2_{Bt}. Consequently, variations in B_g and δ influence the perturbation of the output message, which is subject to approximate noise $e_{\text{out}} \sim \mathcal{N}(0, \sigma^2_{\text{out}})$, in the FHE Activation Function (FHE-ACT).

4 Model Quantization and Fine-Tuning Framework

Our framework transforms pretrained models into quantized models particularly optimized for our FHE-Neuron (§ 3). This quantized model meets the three specific conditions for using FHE-Neuron outlined in § 1.1: (1) computation within the integer domain, (2) adaptation to the limited ciphertext message space, (3) management of approximation errors caused by changes in the precision of neuron computations. As depicted in Figure 6(a), this process initiates with the FHE-aware quantization of the plaintext pretrained model, transitioning it into an integer model and generating scaling factors that comply with the operational range of FHE-Neuron. If this integer model reaches the predefined accuracy threshold during encrypted inference, it is validated and the model is finalized. If not, due to approximation errors, the model undergoes FHE-aware adjustments, where it is fine-tuned to enhance accuracy until it meets the necessary criteria.

4.1 FHE-Aware Quantization

Algorithm 2 Estimate Activation Scaling Factors for FHE-ACT (EstActScale)

Inputs:
- Training data $\mathbf{X} \in \mathbb{Z}^{h \times m}$, where h is the number of features per input sample, and m is the batch size, representing the total number of samples
- Pre-trained model weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$ with n layers,$\mathbf{W_i} \in \mathbb{Z}^{h \times h}$, where k is the number of output features and h is the number of input features per sample
- The message space size of an FHE ciphertext, $P \in \mathbb{Z}$
- Activation Function $f_{\text{ACT}}(x)$ such as ReLU

procedure EstActScale($\mathbf{X}, \mathbf{W}_1, \ldots, \mathbf{W}_n, P$)
  Initialize the sequence of scale factors $\delta_0, \ldots, \delta_{n-1}$ to 1.
  Compute the initial input $\mathbf{Y}_0 = \lfloor \mathbf{X} \cdot \delta_0 \rfloor \in \mathbb{Z}^{h \times m}$.
  for $i = 1$ to n **do**
    $\mathbf{Y}_i = \mathbf{W}_i \cdot \mathbf{Y}_{i-1} \in \mathbb{Z}^{h \times m}$
    ▷Update δ_{i-1}
    $\delta_{i-1} = \begin{cases} \frac{P}{4 \cdot |\max(\mathbf{Y}_i)|} & \text{if } |\max(\mathbf{Y}_i)| \geq |\min(\mathbf{Y}_i)| \\ \frac{P}{4 \cdot |\min(\mathbf{Y}_i)|} & \text{if } |\min(\mathbf{Y}_i)| > |\max(\mathbf{Y}_i)| \end{cases}$
    if $i = 1$ **then**
      $\mathbf{Y}_0 = \lfloor \mathbf{X} \cdot \delta_0 \rfloor \in \mathbb{Z}^{h \times m}$
    end if
    if $i < n$ **then**
      $\mathbf{Y}_i = \mathbf{W}_i \cdot \mathbf{Y}_{i-1} \in \mathbb{Z}^{h \times m}$
      $\mathbf{Y}_i = \lfloor f_{\text{ACT}}(\mathbf{Y}_i) \cdot \delta_i \rfloor$
    end if
  end for
end procedure
Output: Activation quantization scaling factor $\delta_i \in \mathbb{R}, 0 < \delta_i \leq 1, i = 0, \ldots, n-1$

FHE-Aware Quantization is designed to address the constrained message space issue faced in negative cycle lookup table computations with FHE-ACT. When input values exceed the interval from $-\frac{q}{4}$ to $+\frac{q}{4}$, lookup errors or overflows can occur, which may compromise the accuracy of DNN inference. The quantization process is depicted in Figure 6(b), where an Fp32 pre-trained model is transformed into an integer model, with scale factors $\delta_i, i = 0, \ldots, n-1$ assigned for each FHE-ACT. This workflow starts with the post-training quantization (PTQ) of the Fp32 pre-trained model to convert it into an integer model. Subsequently, this integer model is processed through our EstActScale design to calculate the scale

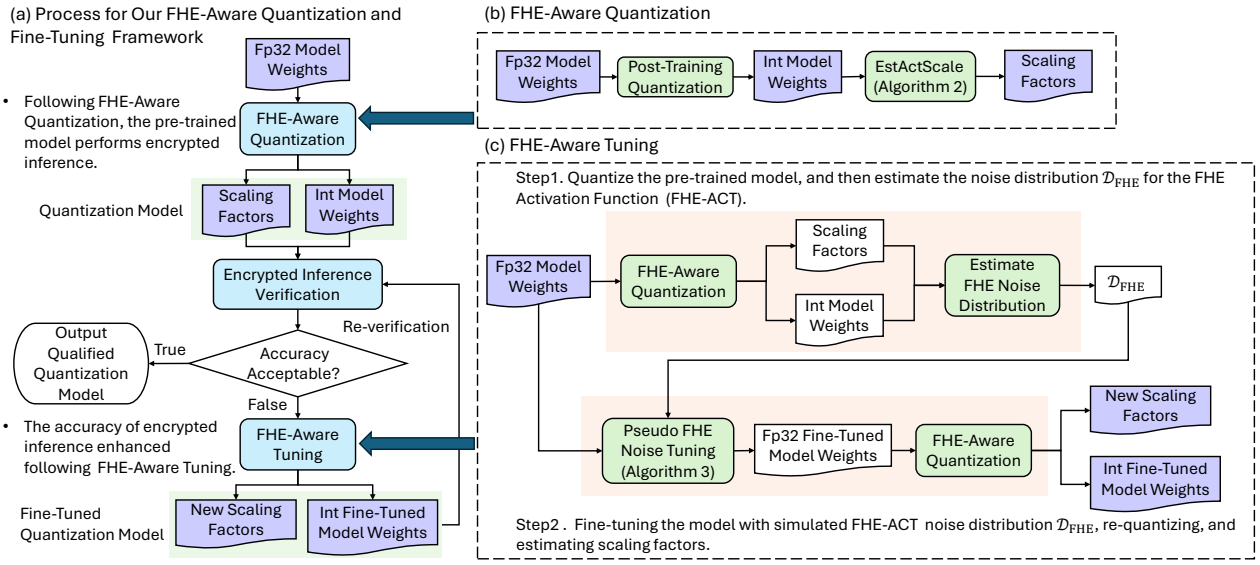

Figure 6: Details of our model Quantization and Fine-tuning Framework

factors, ensuring that the inputs for FHE-ACT stay within the $\pm\frac{q}{4}$ range, thus effectively preventing information overflow during neural computations.

EstActScale (as described in Algorithm 2) calculates the scaling factors for each FHE-ACT within the FHE-DNN framework. It takes as input the training dataset $\mathbf{X} \in \mathbb{Z}^{h\times m}$, quantized pre-trained model weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$ where each layer's weights are represented as $\mathbf{W}_i \in \mathbb{Z}^{h\times h}$, and the message space size $P \in \mathbb{Z}$ of an FHE ciphertext. The algorithm outputs scaling factors $\delta_i \in \mathbb{R}$, which are utilized by the FHE-Neuron (as outlined in Algorithm 1), where each $0 < \delta_i \leq 1$ for $i = 0, \ldots, n-1$.

This algorithm determines the scaling factor δ_{i-1} for the preceding layer's FHE-ACT by calculating the absolute values of the maximum and minimum messages from each layer's linear computations, denoted as $|\max(\mathbf{Y}_i)|$ and $|\min(\mathbf{Y}_i)|$. As outlined in line 8 of Algorithm 2, δ_{i-1} is calculated as the ratio of $\frac{P}{4}$ to the greater of $|\max(\mathbf{Y}_i)|$ and $|\min(\mathbf{Y}_i)|$. This method precisely adjusts the scaling ratio for each layer to ensure that the magnitude of plaintext messages input into the FHE-ACT remains within the $\pm P/4$ range. This range corresponds to the ciphertext input interval of ±q in FHE-ACT, which maps to the plaintext message space of ±P. The scaling factors estimated by EstActScale, in conjunction with the quantized model weights, ensures that the execution of FHE-DNN avoids information overflow.

4.2 FHE-Aware Tuning

We perform FHE-aware model fine-tuning to meticulously mitigate the effects of noise interference e_{out} generated by FHE-ACT on the inference performance of DNNs. FHE-Aware Tuning takes a pre-trained Fp32 model and produces a precisely tuned Int model along with scaling factors. This output model effectively adapts to the noise disturbances caused by FHE-ACT, improving accuracy during encrypted inference. Figure 6 (c) shows that FHE-Aware Tuning is structured in two steps:

First Step: This process starts by inputting a Fp32 pre-trained model, a set of FHE parameters, and corresponding training data. The main goal is to derive the approximate noise distribution $\mathcal{D}_{\mathrm{FHE}}$ for each FHE-ACT in the model. Initially, the pre-trained model converts into an integer model through FHE-Aware Quantization, and we precisely estimate the necessary scaling factors $\delta_i, i = 0, \ldots, n-1$ for each FHE-ACT. Ultimately, using Eq. (5), we estimate the FHE-ACT noise distribution $\mathcal{D}_{\mathrm{FHE}}$, which we use to fine-tune the model in subsequent steps.

Second Step: This process employs the same inputs as the initial step, utilizing the previously obtained $\mathcal{D}_{FHE}$ noise distribution. Its objective is to refine a model to significantly reduce the detrimental effects of FHE-ACT noise on DNN performance. For this purpose, we have developed a method called Pseudo FHE-Noise Tuning (see Algorithm 3), which fine-tunes Fp32 pretrained models to accommodate the noise characteristics of $\mathcal{D}_{FHE}$. Following this, we apply FHE-Aware Quantization to transform the fine-tuned float32 model into an integer model. We then re-evaluate and adjust the scaling factors to prevent computational overflow. These modifications ensure that the model's accuracy during encrypted inference substantially surpasses its performance prior to adjustment.

Our Pseudo FHE-Noise Tuning model fine-tuning method is detailed in Algorithm 3, building upon the foundational Pseudo-Noise Tuning approach presented in Algorithm 4 in Section § B. This algorithm accepts several inputs: training data $\mathbf{X} \in \mathbb{R}^{h\times m}$, a label matrix $\mathbf{T} \in \mathbb{R}^{q\times m}$, pre-trained model weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$ (where each layer's weights $\mathbf{W}_i$ are in $\mathbb{R}^{h\times h}$, for a total of n layers), quantization scaling factors for activation functions $\delta_i \in \mathbb{R}$, where $0 < \delta_i \leq 1$ for $i = 0, \ldots, n-1$, the message space size $P \in \mathbb{Z}$ for the FHE ciphertexts, and the noise distribution $\mathcal{D}_{\mathrm{FHE}}$ associated with FHE-ACT. The output includes the fine-tuned model weights $\mathbf{W}_1^*, \mathbf{W}_2^*, \ldots, \mathbf{W}_n^*$, which enhance the accuracy of the model on encrypted inference compared to its performance prior to fine-tuning.

Based on the analysis of FHE-ACT in §3.2 for the noise distribution $\mathcal{D}_{\mathsf{FHE}}$, we perform model finetuning as in Algorithm 3. Additionally, we designed a noise proxy function f_{FHE} according to the characteristics of the FHE-Neuron. The corresponding formula is as follows:

$$e \sim \mathcal{D}_{\mathsf{FHE}} = \mathcal{N}(0, \sigma^2_{\text{out}}) \frac{P}{Q_{in}} \in \mathbb{Z}_P \text{ and} \tag{5}$$

$$f_{\mathsf{FHE}}(\mathbf{Y} \mid P, \delta, e) = (\frac{|\max(\mathbf{Y}) - \min(\mathbf{Y})|}{P \cdot \delta} e) + \mathbf{Y} \in \mathbb{R}. \tag{6}$$

Eq. (5) describes the distribution $\mathcal{D}_{\mathsf{FHE}}$ which is derived from the description of σ^2_{out} in Eq. (4) found in Section 3.2. Eq. (6) outlines the Noise Proxy function of the FHE-ACT. This function scales the pseudo FHE-Noise e, which is within the integer domain, according to the FHE message space P and the scaling factor δ of the FHE-ACT, mapping it to the real-number interval for inputs into the $f_{\text{ACT}}(\cdot)$ function during model fine-tuning, as shown in Figure 7. In § 5, we further demonstrate that FHE-aware model fine-tuning improves the inference accuracy of DNNs. For example, Table 5 illustrates that in the cifar10 experiments, fine-tuning the FHE parameters can increase the model inference accuracy by 15%.

We note that this method is applicable not only to third-generation FHE but also to other FHE schemes for DNN fine-tuning. For instance, the CKKS scheme uses approximate computation, similar to our approach. We further discuss these extensions in § 6.

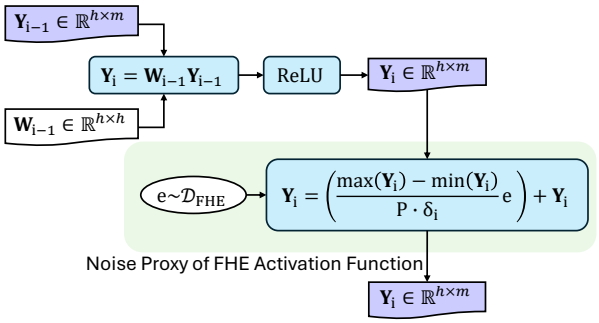

Figure 7: Model Fine-tuning using simulated FHE Activation Function (FHE-ACT) noise

Algorithm 3 Pseudo FHE-Noise Tuning

Inputs:
- Training data $\mathbf{X} \in \mathbb{R}^{h \times m}$, where h is the number of features per input sample, and m is the batch size, representing the total number of samples.
- The label matrix $\mathbf{T} \in \mathbb{R}^{q \times m}$ where q is the number of classes for multi-class classification and m matches the batch size in $\mathbf{X}$.
- Pre-trained model weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$ with n layers,$\mathbf{W_i} \in \mathbb{R}^{h \times h}$, where h is the number of output features and h is the number of input features per sample.
- Activation quantization scaling factor $\delta_i \in \mathbb{R}, 0 < \delta_i \leq 1, i = 0, \ldots, n-1$.
- The message space size of an FHE ciphertext, $P \in \mathbb{Z}$.
- The noise distribution $\mathcal{D}_{\mathsf{FHE}}$ of FHE-ACT.
- Activation Function $f_{\text{ACT}}(x)$ such as ReLU

procedure FineTuning($\mathbf{X}, \mathbf{T}, \mathbf{W}_1, \ldots, \mathbf{W}_n, \delta_0, \ldots, \delta_{n-1}, \sigma, P$)
  Initialize the input $\mathbf{Y}_0 = \mathbf{X}$, and new weights $\mathbf{W}^*_1, \mathbf{W}^*_2, \ldots, \mathbf{W}^*_n$ to $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$.
  for $i = 1$ to n **do**
    $\mathbf{Y}_i = \mathbf{W}^*_i \cdot \mathbf{Z}_{i-1}, \mathbf{Y}_i \in \mathbb{R}^{h \times m}$
    $\mathbf{Y}_i = f_{\text{ACT}}(\mathbf{Y}_i)$
    ▷ Start Noise Proxy
    $e \sim \mathcal{D}_{\mathsf{FHE}}$
    $\mathbf{Y}_i = f_{\mathsf{FHE}}(\mathbf{Y}_i \mid P, \delta, e)$
    ▷ End Noise Proxy
  end for
  Compute loss $L(\mathbf{Y}_n, \mathbf{T})$ using Cross-Entropy Loss
  Update weights $\mathbf{W}^*_1, \ldots, \mathbf{W}^*_n$ using backpropagation with SGD
end procedure
Outputs: Fine-Tuned model weights $\mathbf{W}^*_1, \mathbf{W}^*_2, \ldots, \mathbf{W}^*_n$ with n layers, $\mathbf{W}^*_i \in \mathbb{R}^{k \times h}$.

5 Evaluation

We perform a comprehensive evaluation and comparison of our method against several state-of-the-art works. § 5.1 outlines the experimental settings, including the FHE parameters, hardware, software,the datasets used, and the network architecture. § 5.2 presents the validation results for the MNIST and Fashion-MNIST datasets, and § 5.3 describes the validation results for the CIFAR-10 dataset. Tables 3 and 4 present a comparative evaluation of our work with previous efforts.

5.1 Experimental Settings

Hardware and Software: We implement Encrypted DNN model inference using the NTT-based OpenFHE library [2] version 1.0.3 and the FFT-based GPU-accelerated FHEW library [54]. We conduct experiments on ten different machine configurations, including systems with various numbers of RTX 3090, RTX 4090, A100, and H100 GPUs. We provide details of these configurations in Table 9 in Section A. In the experiments on these ten computing platforms, less than 20 GB of memory is used. The operating system is Ubuntu version 20.04.6 LTS.

FHE parameters: The parameters for the FHEW were meticulously configured in accordance with [37] to ensure a security level of 128 bits. The specific settings included a message space size of $P = 2^{16}$, an input LWE ciphertext modulus of $Q_{in} = 2^{35}$, and a ring dimension for the LWE scheme at $n = 1,305$. Additionally, the ring dimension for RLWE/RGSW was set at $N = 2,048$, with an RLWE/RGSW modulus (utilized for NTTs or FFTs) of $Q = 2^{54}$. The modulus of the LWE ciphertext for bootstrapping was set to $q = 4096$, and the gadget base used for key switching was $B_{ks} = 2^7$.

The gadget base for digit decomposition, B_g, directly impacts the noise level of the FHE-Neuron. A smaller B_g results in less noise but increases the computation time. During our experiments, we access the impacts of $B_g = 2^{27}, 2^{18}, 2^9$ on the accuracy and performance of DNN inference. FHE-ACT using NTT or FFT impacts noise. As

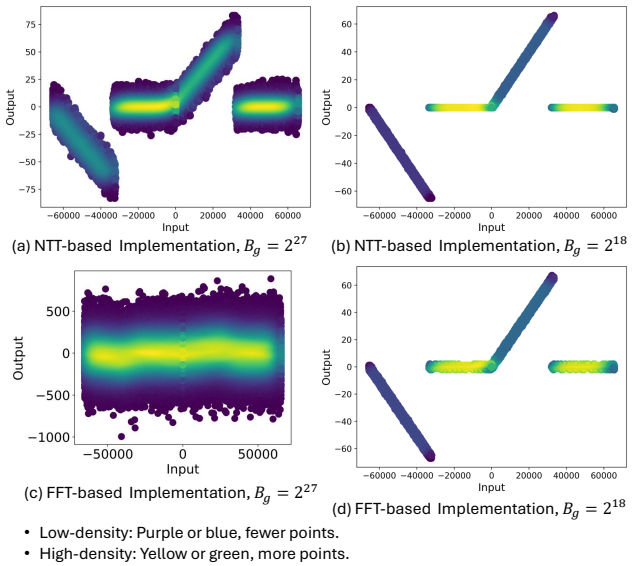

Figure 8: Compare the noise variations in the outputs. The comparison is conducted at $B = 2^{27}$ and $B = 2^{18}$ between two different FHE-ACT (ReLU) implementations, one based on NTT and the other on FFT, using a scaling factor of $\delta = 0.002$.

shown in Figure 8, FFT-based FHE-ACT with $B_g = 2^{27}$ causes excessive noise, making it unsuitable for DNN inference. To minimize noise, B_g should not exceed 2^{18}.

Benchmark and dataset: Our datasets, which include MNIST [30], Fashion MNIST [53], and CIFAR-10 [27], are commonly used as benchmarks in the field of computer vision. Each dataset comprises images across 10 categories.

Network architecture: We summarize the network architectures evaluated on various datasets in Appendix (Tables 10 and 11 in § A). For the grayscale MNIST and Fashion MNIST datasets, we experimented with a simple two-layer fully connected neural network to explore the performance of the MNIST dataset under the same model architecture as EFHEPENN [29] and FHE-DiNN [4]. For the colored CIFAR-10 dataset, we used four types of VGG neural networks [49] for experiments.

Hyperparameter Settings for Model Fine-Tuning: Our model fine-tuning uses Stochastic Gradient Descent (SGD) with a momentum of 0.9 and a weight decay of 5×10^{-4} to enhance stability and prevent overfitting. Training runs for 150 epochs with a batch size of 128 and an initial learning rate of 0.01. A StepLR scheduler reduces the learning rate by 90% every 50 epochs.

5.2 Results on MNIST and Fashion-MNIST

Our method consistently outperforms known (to our knowledge) third-generation FHE (FHEW/TFHE) solutions in terms of average computation time. As highlighted in Table 3, our experimental results surpass those of similar GPU-accelerated technologies, such as REDsec [17] and PBE [11], as well as CPU-based approaches like EFHEPENN [29], FDFB [26], and FHE-DiNN [4]. Notably, our method achieves nearly plaintext-level accuracies on MNIST and Fashion-MNIST, with scores of 96.86% and 89% respectively, without the need for FHE-Aware Tuning.

Table 3: Comparison with Prior Works on MNIST and Fashion-MNIST. The symbol † marks experiments conducted using an Intel Xeon Processor (Icelake) CPU, 1.23 TB RAM and seven NVIDIA H100 GPUs.

Method	Eval	Dataset	Scheme	FHE Acc.	Time	Sec.
Ours†	GPU	F-MNIST	FHEW	88.42%	17 ms	128-bit
Ours†		MNIST	FHEW	96.52%	4.5 ms	128-bit
REDsec [17]		MNIST	TFHE	99%	8.2 s	128-bit
PBE [11]		MNIST	TFHE	97.1%	7.53 s	128-bit
EFHEPENN [29]	CPU	MNIST	FHEW	94.04%	140 ms	80-bit
REDsec [17]		MNIST	TFHE	99%	18.4 s	128-bit
FDFB [26]		MNIST	FHEW	95%	2736 s	100-bit
FHE-DiNN [4]		MNIST	TFHE	93.71%	490 ms	80-bit

Table 4: Comparative Analysis on CIFAR-10. The symbols † is described in Table 3. Note that REDsec [17], DaCapo [9], SHE [38], and TFHE-DNN[51] do not provide information on memory usage in their papers.

Method	Eval	Scheme	FHE Acc.	Time	Memory
Ours†	GPU	FHEW	90.5%	30 s	19.6 GB
REDsec [17]		TFHE	88.5%	1769.4 s	-
DaCapo [9]		CKKS	93.8%	53 s	-
OPP-CNN [24]	CPU	CKKS	94.12%	544 s	100 GB
PPML-DNN [31]		CKKS	92.43%	10,602 s	176 GB
EVA [13]		CKKS	79.34%	72.7s	190 GB
Falcon [39]		BFV	76.5%	107 s	32 GB
LoLa [6]		BFV	74.1%	730 s	12 GB
SHE [38]		TFHE	94.62%	12,041 s	-
TFHE-DNN [51]		TFHE	87.5%	18,000 s	-

Table 5: Evaluation on CIFAR-10 Using CPU (1,000 test samples).

Network	PL Acc.	B_g	Fine-tuned	FHE Acc.	Time
VGG6-32	85.6%	2^{27}	Yes	75.6%	638 s
		2^{27}	No	58%	638 s
		2^{18}	Yes	85%	803 s
		2^{18}	No	81.8%	803 s
VGG6-24	83.8%	2^{27}	Yes	75%	476 s
		2^{27}	No	60%	476 s
		2^{18}	Yes	83%	617 s
		2^{18}	No	76.8%	617 s

5.3 Results on CIFAR-10

The evaluation of the CIFAR-10 dataset provides a comprehensive insight into the robustness and scalability of our method under complex task conditions. Tables 4 compares different methods on the CIFAR-10 dataset in terms of FHE accuracy, time, and memory usage. Our FHE-DNN approach achieved 90.5% accuracy using a GPU, slightly lower than the 93.8% of DaCapo [9] (CKKS) but higher

Table 6: Evaluation on CIFAR-10 Using GPU. Only the VGG9 experiment uses 10,000 test samples; all others use 5,000 samples. Experiments are performed using an Intel Xeon Processor (Icelake) CPU, 1.23 TB RAM and seven NVIDIA H100 GPUs.

Network	PL Acc.	B_g	Fine-tuned	FHE Acc.	Time
VGG9	91.5%	2^{18}	Yes	90.5%	30 s
		2^{18}	No	85.8%	30 s
VGG6-96	86.5%	2^{18}	Yes	86%	16 s
		2^{18}	No	78%	16 s
		2^{9}	No	79%	26 s
VGG6-32	85.6%	2^{18}	Yes	84.2%	6 s
		2^{18}	No	79.2%	6 s
		2^{9}	No	80.5%	10 s
VGG6-24	83.7%	2^{18}	Yes	82.35%	5 s
		2^{18}	No	78.2%	5 s
		2^{9}	No	80.7%	8 s

than the 88.5% of REDsec [17]. Our inference time is a mere 30 seconds, dramatically faster than REDsec's (TFHE) 1769.4 seconds, and comparable to DaCapo's 53 seconds. Moreover, our method only utilizes 19.6 GB of memory, a stark contrast to the 176 GB and 190 GB required by PPML-DNN [31] and EVA [13], respectively, both of which use the CKKS scheme. This showcases our significant advantage in memory efficiency. Note that REDsec and DaCapo did not report memory usage. Overall, our GPU-accelerated method is significantly faster than other third-generation FHE methods and uses notably less memory compared to CKKS-based methods.

5.4 Our Method's Verification in Various GPUs

To fairly compare our work with previous studies and facilitate future researchers in benchmarking our results, we provide experimental data obtained from testing our method on systems with different GPU models and numbers of GPUs, as shown in Table 7. On an RTX 3090 GPU, the average processing times for MNIST and CIFAR-10 images using our method are 104 ms and 765 s, respectively. This outperforms REDsec [17], which also uses third-generation FHE but runs on 8 NVIDIA T4 GPUs, with processing times of 8.2 s and 1769.4 s for the same tasks. Additionally, our method achieves a faster MNIST processing time (104 ms) compared to PBE [11], which uses 8 NVIDIA A100 GPUs and records a processing time of 7.53 s.

Although our method is still slower than the CKKS-based DaCapo [9] under RTX 3090 GPU, our experimental results highlight strong potential of our approach towards practical deployments, with enhanced computational power and further optimizations.

5.5 Gadget Base Size B_g Balances Accuracy and Time in Encrypted DNN Inference

The gadget base size B_g plays a key role in encrypted DNN inference, affecting both accuracy and computation time. Reducing B_g can improve accuracy, but it also increases computation time (see Table 5). However, when B_g drops below 2^{18}, the accuracy gains become minimal, and the additional computation time is not worthwhile (see Table 6).

Our **FHE-aware fine-tuning** method improves encrypted DNN accuracy without adding extra computation time. As shown in Table 5, with $B_g = 2^{27}$, a single fine-tuning session boosts the accuracy of VGG6-32 and VGG6-24 by 17.6% and 15%, respectively. Additionally, Table 6 shows that with $B_g = 2^{18}$, the fine-tuned model achieves encrypted inference accuracy close to plaintext inference.

5.6 Activation Functions as Performance Bottlenecks in Encrypted DNN Inference

For applications using third-generation FHE, Bootstrapping often becomes a performance bottleneck. We design an FHE-Neuron with adjustable ciphertext precision to reduce the reliance on Bootstrapping. However, Bootstrapping still remains the main source of time consumption. For example, in experiments with the CIFAR-10 dataset on four NVIDIA RTX 4090 GPUs, the time for all models to perform linear computations does not exceed 2 seconds (see Tables 12 in Appendix § A), but most of the time is spent during the Bootstrapping stage of the FHE activation functions. Therefore, speeding up Bootstrapping through hardware or algorithm improvements, or reducing the use of activation functions in the model, could help enhance the performance of encrypted inference.

5.7 Time Cost of FHE-Aware Quantization and Fine-Tuning

Table 8 summarizes the execution time of the proposed **FHE-aware quantization** (Algorithm 2) and **FHE-aware fine-tuning** (Algorithm 3) across various datasets and network architectures. All experiments were conducted on a workstation equipped with an AMD Ryzen Threadripper PRO 5975WX (32 cores, 64 threads) and an NVIDIA RTX 4090 GPU. The quantization stage is performed entirely on the CPU, estimating quantization parameters using a randomly sampled subset of 128 training examples.

The fine-tuning stage operates over the full training dataset. We report the time cost for 50 training epochs. As shown in Table 8, the cost of fine-tuning is comparable to that of standard training, and in most cases, 50 epochs of fine-tuning are sufficient to compensate for the inference degradation caused by encryption-induced noise. Crucially, both quantization and fine-tuning are offline, one-time procedures conducted during model preparation, and thus do not affect the runtime latency during encrypted inference.

For lightweight datasets such as MNIST and FashionMNIST, the impact of FHE-induced noise on inference accuracy is minimal. Consequently, quantization alone is typically sufficient to maintain high accuracy, and fine-tuning can often be skipped entirely.

6 Extending Our Method to Other FHE Schemes

Our computing paradigm significantly improves efficiency in third-generation FHE for DNN encrypted inference. We also observe that the underlying ideas may be extended to other FHE schemes. Below, we illustrate this potential using the CKKS scheme as an example.
Neuron Design: CKKS-based DNNs typically rely on larger FHE parameters to tolerate noise throughout the computation, which results in substantial computational, memory, and storage overhead. By applying the design principles of FHE-Neuron (see § 3), it

Table 7: Average Computation Time per Image for Encrypted DNN Execution on Different GPU Server Configurations. The results indicate model performance across various network architectures with the FHE parameter B_g.

Dataset	Network	B_g	RTX 3090 x1	RTX 4090 x1	RTX 4090 x2	RTX 4090 x4	RTX 4090 x12	A100 x1	H100 x1	H100 x4	H100 x7
MNIST	Net-30	2^{18}	104 ms	43 ms	24 ms	12.6 ms	5.8 ms	33 ms	23 ms	8 ms	4.5 ms
FashionMNIST	Net-128	2^{18}	440 ms	187 ms	95 ms	50 ms	21 ms	129 ms	90 ms	24 ms	17 ms
CIFAR-10	VGG9	2^{18}	765 s	325 s	162 s	85 s	34 s	217 s	146 s	43 s	30 s
		2^{9}	1,738 s	793 s	400 s	206 s	77 s	473 s	323 s	89 s	48 s
CIFAR-10	VGG6-96	2^{18}	436 s	186 s	91 s	49 s	20 s	126 s	89 s	25 s	16 s
		2^{9}	988 s	464 s	227 s	117 s	43 s	274 s	186 s	52 s	26 s
CIFAR-10	VGG6-32	2^{18}	147 s	61 s	32 s	17 s	8 s	43 s	29 s	9 s	6 s
		2^{9}	338 s	154 s	78 s	40 s	16 s	91 s	62 s	18 s	10 s
CIFAR-10	VGG6-24	2^{18}	112 s	45 s	25 s	13 s	6 s	33 s	23 s	6 s	5 s
		2^{9}	254 s	119 s	59 s	30 s	12 s	69 s	47 s	13 s	8 s

Table 8: Timing Analysis of FHE-Aware Quantization and Fine-Tuning across Different Datasets and Neural Architectures. Quant. denotes FHE-aware quantization, executed solely on the CPU. Both training and fine-tuning are conducted for 50 epochs.

Dataset	Network	Quant.	Training	Tuning
MNIST	Net-30	0.44 s	249 s	270 s
FashionMNIST	Net-128	2.73 s	275 s	286 s
CIFAR-10	VGG9	1,301 s	4,474 s	4,683 s
CIFAR-10	VGG6-96	438 s	2,504 s	2,675 s
CIFAR-10	VGG6-32	28 s	1,188 s	1,265 s
CIFAR-10	VGG6-24	19 s	1,108 s	1,129 s

becomes feasible to use smaller FHE parameters while maintaining similar accuracy, thereby reducing computational costs. We elaborate on this approach below.

Model Quantization: Reducing FHE parameters in CKKS-DNNs naturally decreases the precision of the underlying plaintext, increasing the risk of overflow. Our quantization method (see § 4.1) directly addresses this issue by converting models from full precision (e.g., FP32) to low precision (e.g., FP4), while carefully estimating appropriate Activation Scaling Factors. This enables stable and reliable inference performance in encrypted settings.

Model Fine-Tuning: Lowering FHE parameters in CKKS makes the system less error-tolerant, amplifying the impact of relative error during computation and potentially degrading inference accuracy. Our fine-tuning strategy (see § 4.2) mitigates this by first conducting a theoretical analysis to identify key error sources specific to the FHE (e.g., CKKS in this example) computations. Guided by this analysis, we fine-tune the model weights to maintain high inference accuracy, even in the presence of scheme-specific noise characteristics.

We leave concrete optimization of parameters and performances (for CKKS and others) as an interesting research direction.

7 Conclusion

This work advances the practicality of third-generation Fully Homomorphic Encryption (FHE) for deep neural network (DNN) inference by addressing the high computational costs that have limited its real-world applicability. We propose FHE-Neuron, an optimized encrypted neural architecture that dynamically adjusts ciphertext precision to balance efficiency and accuracy. To further enhance performance, we introduce an FHE-Aware Quantization and Fine-Tuning Framework, which adapts pre-trained models for encrypted inference by optimizing precision and mitigating noise-related errors. Our extensive experiments on MNIST, Fashion MNIST, and CIFAR-10 demonstrate that our method significantly reduces inference latency while maintaining high accuracy, outperforming previous FHE-based approaches. By leveraging the efficiency of FHEW/TFHE with precision-switching strategies, our approach paves the way for more scalable, cost-effective encrypted deep learning solutions.

Future research can explore several key directions to further enhance the practicality and efficiency of encrypted deep learning. First, extending our approach to a broader range of model architectures, such as Transformers and Graph Neural Networks (GNNs), could enable privacy-preserving computations in more complex tasks, including natural language processing and graph-based data analysis. Second, integrating our method with advanced performance optimization techniques in third-generation FHE, such as Single Instruction Multiple Data (SIMD), could significantly improve computational efficiency by enabling parallel encrypted operations. These advancements would not only accelerate encrypted inference but also broaden the applicability of FHE-based deep learning across diverse domains, including secure federated learning, biomedical data analysis, and encrypted financial modeling.

Acknowledgments

The content of this paper is entirely the original work of the authors, with only the English grammar polished by ChatGPT4. This work was primarily conducted at Inventec Corporation. We gratefully acknowledge the support of the National Science and Technology Council, Taiwan, under grant NSTC 112-2221-E-002 -159 -MY3 and 113-2634-F-002-001 -MBK. Feng-Hao Liu would like to thank NSF Career Award CNS-2402031. I-Ping Tu would like to thank Academia Sinica Investigator Award AS-IA-110-M05. Their support was instrumental in developing critical preliminary results for this work.

References

[1] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On The Concrete Hardness Of Learning With Errors. Cryptology ePrint Archive, Report 2015/046. https://eprint.iacr.org/2015/046

[2] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R. V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Report 2022/915. https://eprint.iacr.org/2022/915

[3] Maya Bakshi and Mark Last. 2020. CryptoRNN - Privacy-Preserving Recurrent Neural Networks Using Homomorphic Encryption. In *CSCML*, Vol. 12161. Springer, 245–253. https://doi.org/10.1007/978-3-030-49785-9_16

[4] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). 483–512. https://doi.org/10.1007/978-3-319-96878-0_17

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory* 6, 3 (2014), 13:1–13:36. https://doi.org/10.1145/2633600

[6] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low Latency Privacy Preserving Inference. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 812–821. http://proceedings.mlr.press/v97/brutzkus19a.html

[7] Zhicheng Cai and Chenglei Peng. 2021. A study on training fine-tuning of convolutional neural networks. In *13th International Conference on Knowledge and Smart Technology, KST 2021, Bangsaen, Chonburi, Thailand, January 21-24, 2021*. IEEE, 84–89. https://doi.org/10.1109/KST51265.2021.9415793

[8] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT 2017, Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[9] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Suncheul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. 2024. DaCapo: Automatic Bootstrapping Management for Efficient Fully Homomorphic Encryption. In *USENIX Security Symposium*. USENIX Association.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016, Part I (LNCS, Vol. 10031)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). 3–33. https://doi.org/10.1007/978-3-662-53887-6_1

[11] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12716)*. Springer, 1–19. https://doi.org/10.1007/978-3-030-78086-9_1

[12] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2021. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In *ASIACRYPT 2021, Part III (LNCS, Vol. 13092)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). 670–699. https://doi.org/10.1007/978-3-030-92078-4_23

[13] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. 2020. EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*. ACM, 546–561. https://doi.org/10.1145/3385412.3386023

[14] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *ACM SIGPLAN PLDI*. ACM, 142–156. https://doi.org/10.1145/3314221.3314628

[15] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). 617–640. https://doi.org/10.1007/978-3-662-46800-5_24

[16] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* (2012), 144. http://eprint.iacr.org/2012/144

[17] Lars Folkerts, Charles Gouert, and Nektarios Georgios Tsoutsos. 2023. REDsec: Running Encrypted Discretized Neural Networks in Seconds. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/redsec-running-encrypted-discretized-neural-networks-in-seconds/

[18] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NeurIPS*. 2672–2680.

[20] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. http://arxiv.org/abs/1510.00149

[21] Ming-Chien Ho, Yu-Te Ku, Yu Xiao, Feng-Hao Liu, Chih-Fan Hsu, Ming-Ching Chang, Shih-Hao Hung, and Wei-Chao Chen. 2024. Invited Paper: Efficient Design of FHEW/TFHE Bootstrapping Implementation with Scalable Parameters. In *2024 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*.

[22] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *SIGSAC*. ACM, 1209–1222. https://doi.org/10.1145/3243734.3243837

[23] Leonardo Rezende Juracy, Rafael Garibotti, and Fernando Gehm Moraes. 2023. From CNN to DNN Hardware Accelerators: A Survey on Design, Exploration, Simulation, and Frameworks. *Found. Trends Electron. Des. Autom.* 13, 4 (2023), 270–344. https://doi.org/10.1561/1000000060

[24] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* 18 (2023), 2175–2187. https://doi.org/10.1109/TIFS.2023.3263631

[25] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[26] Kamil Kluczniak and Leonard Schild. 2021. FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2021/1135. https://eprint.iacr.org/2021/1135

[27] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. , 32–33 pages. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf U. Toronto Tech Report.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*. 1106–1114.

[29] Kwok-Yan Lam, Xianhui Lu, Linru Zhang, Xiangning Wang, Huaxiong Wang, and Si Qi Goh. 2023. Efficient FHE-based Privacy-Enhanced Neural Network for AI-as-a-Service. Cryptology ePrint Archive, Paper 2023/647. https://eprint.iacr.org/2023/647 https://eprint.iacr.org/2023/647.

[30] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[31] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2021. Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network. Cryptology ePrint Archive, Report 2021/783. https://eprint.iacr.org/2021/783

[32] Feng-Hao Liu and Han Wang. 2023. Batch Bootstrapping I: A New Framework for SIMD Bootstrapping in Polynomial Modulus. In *EUROCRYPT 2023, Part III (LNCS)*. 321–352. https://doi.org/10.1007/978-3-031-30620-4_11

[33] Feng-Hao Liu and Han Wang. 2023. Batch Bootstrapping II: Bootstrapping in Polynomial Modulus only Requires $\tilde{O}(1)$ FHE Multiplications in Amortization. In *EUROCRYPT 2023, Part III (LNCS)*. 353–384. https://doi.org/10.1007/978-3-031-30620-4_12

[34] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C. Kot. 2020. NTU RGB+D 120: A Large-Scale Benchmark for 3D Human Activity Understanding. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 10 (2020), 2684–2701. https://doi.org/10.1109/TPAMI.2019.2916873

[35] Tzu-Li Liu, Yu-Te Ku, Ming-Chien Ho, Feng-Hao Liu, Ming-Ching Chang, Chih-Fan Hsu, Wei-Chao Chen, and Shih-Hao Hung. 2023. An Efficient CKKS-FHEW/TFHE Hybrid Encrypted Inference Framework. In *Computer Security. ESORICS 2023 International Workshops - CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25-29, 2023, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 14399)*. Springer, 535–551. https://doi.org/10.1007/978-3-031-54129-2_32

[36] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. 2021. Large-Precision Homomorphic Sign Evaluation using FHEW/TFHE Bootstrapping. Cryptology ePrint Archive, Report 2021/1337. https://eprint.iacr.org/2021/1337

[37] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. 2022. Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping. In *ASIACRYPT 2022, Part II (LNCS, Vol. 13792)*, Shweta Agrawal and Dongdai Lin (Eds.). 130–160. https://doi.org/10.1007/978-3-031-22966-4_5

[38] Qian Lou and Lei Jiang. 2019. SHE: A Fast and Accurate Deep Neural Network for Encrypted Data. In *NeurIPS*. 10035–10043.

[39] Qian Lou, Wen-jie Lu, Cheng Hong, and Lei Jiang. 2020. Falcon: Fast Spectral Inference on Encrypted Data. In *Advances in Neural Information Processing Systems*

33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. https://proceedings.neurips.cc/paper/2020/hash/18fc72d8b8aba03a4d84f66efabce82e-Abstract.html

[40] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj. 2022. Survey on Fully Homomorphic Encryption, Theory, and Applications. Cryptology ePrint Archive, Report 2022/1602. https://eprint.iacr.org/2022/1602

[41] Daniele Micciancio and Yuriy Polyakov. 2021. Bootstrapping in FHEW-like Cryptosystems. In *WAHC*. ACM, 17–28. https://doi.org/10.1145/3474366.3486924

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[43] Hongwu Peng, Ran Ran, Yukui Luo, Jiahui Zhao, Shaoyi Huang, Kiran Thorat, Tong Geng, Chenghong Wang, Xiaolin Xu, Wujie Wen, and Caiwen Ding. 2023. LinGCN: Structural Linearized Graph Convolutional Network for Homomorphically Encrypted Inference. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/41bd71e7bf7f9fe68f1c936940fd06bd-Abstract-Conference.html

[44] Robert Podschwadt and Daniel Takabi. 2020. Classification of Encrypted Word Embeddings using Recurrent Neural Networks. In *PrivateNLP WSDM (CEUR Workshop Proceedings, Vol. 2573)*. CEUR-WS.org, 27–31. https://ceur-ws.org/Vol-2573/PrivateNLP_Paper3.pdf

[45] Robert Podschwadt and Daniel Takabi. 2021. Non-interactive Privacy Preserving Recurrent Neural Network Prediction with Homomorphic Encryption. In *CLOUD*. IEEE, 65–70. https://doi.org/10.1109/CLOUD53861.2021.00019

[46] Ran Ran, Wei Wang, Quan Gang, Jieming Yin, Nuo Xu, and Wujie Wen. 2022. CryptoGCN: Fast and Scalable Homomorphically Encrypted Graph Convolutional Network Inference. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. http://papers.nips.cc/paper_files/paper/2022/hash/f5332c8273d02729730a9c24dec2135e-Abstract-Conference.html

[47] Ran Ran, Nuo Xu, Tao Liu, Wei Wang, Gang Quan, and Wujie Wen. 2023. Penguin: Parallel-Packed Homomorphic Encryption for Fast Graph Convolutional Network Inference. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/3cc685788a311fa35d8d41df93e288ca-Abstract-Conference.html

[48] Juncheol Shin, Junhyuk So, Sein Park, Seungyeop Kang, Sungjoo Yoo, and Eunhyeok Park. 2023. NIPQ: Noise proxy-based Integrated Pseudo-Quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 3852–3861. https://doi.org/10.1109/CVPR52729.2023.00375

[49] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*. http://arxiv.org/abs/1409.1556

[50] Daniel Sonntag, Michael Barz, Jan Zacharias, Sven Stauden, Vahid Rahmani, Áron Fóthi, and András Lörincz. 2017. Fine-tuning deep CNN models on specific MS COCO categories. *CoRR* abs/1709.01476 (2017). arXiv:1709.01476 http://arxiv.org/abs/1709.01476

[51] Andrei Stoian, Jordan Fréry, Roman Bredehoft, Luis Montero, Celia Kherfallah, and Benoît Chevallier-Mames. 2023. Deep Neural Networks for Encrypted Inference with TFHE. In *Cyber Security, Cryptology, and Machine Learning - 7th International Symposium, CSCML 2023, Be'er Sheva, Israel, June 29-30, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13914)*. Springer, 493–500. https://doi.org/10.1007/978-3-031-34671-2_34

[52] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *NeurIPS*. 3104–3112.

[53] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017). arXiv:1708.07747 http://arxiv.org/abs/1708.07747

[54] Yu Xiao, Feng-Hao Liu, Yu-Te Ku, Ming-Chien Ho, Chih-Fan Hsu, Ming-Ching Chang, Shih-Hao Hung, and Wei-Chao Chen. 2025. GPU Acceleration for FHEW/TFHE Bootstrapping. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025, 1, 314–339.

[55] Xun Zhou, A. Kai Qin, Maoguo Gong, and Kay Chen Tan. 2021. A Survey on Evolutionary Construction of Deep Neural Networks. *IEEE Trans. Evol. Comput.* 25, 5 (2021), 894–912. https://doi.org/10.1109/TEVC.2021.3079985

A Experimental Settings and Detail Supplement

This section supplements the experimental details and settings. Tables 9, 10, 11 and 12 provide supplemental information to the experimental settings and results. Table 9 details the four types of machine specifications used in our experiments. Tables 10 and 11 describe the model architectures and corresponding datasets used in the experiments. Finally, We conduct a detailed analysis of the computation times for linear calculations and activation functions during the encrypted inference process in Table 12 to identify performance bottlenecks.

Table 9: Experiment Computing Platforms Specifications

Platform Type	CPU	Memory	GPU
CPU-only	AMD Ryzen Threadripper PRO 5975WX (32 cores, 64 threads)	256GB	None
RTX 3090 x1	AMD EPYC 7552 48-Core Processor	256GB	One NVIDIA RTX 3090
RTX 4090 x1	AMD Ryzen Threadripper PRO 5975WX (32 cores, 64 threads)	256GB	One NVIDIA RTX 4090
RTX 4090 x2	AMD Ryzen Threadripper PRO 5975WX (32 cores, 64 threads)	256GB	Two NVIDIA RTX 4090
RTX 4090 x4	AMD Ryzen Threadripper PRO 5975WX (32 cores, 64 threads)	256GB	Four NVIDIA RTX 4090
RTX 4090 x12	AMD Ryzen Threadripper PRO 5995WX (64 cores, 128 threads)	378GB	Twelve NVIDIA RTX 4090
A100 x1	AMD EPYC 7643 (48 cores, 96 threads)	1.008TB	One NVIDIA A100
H100 x1	Intel Xeon Processor (Icelake) (24 cores, 48 threads)	1.23TB	One NVIDIA H100
H100 x4	Intel Xeon Processor (Icelake) (24 cores, 48 threads)	1.23TB	Four NVIDIA H100
H100 x7	Intel Xeon Processor (Icelake) (24 cores, 48 threads)	1.23TB	Seven NVIDIA H100

B Pseudo-Noise Tuning

The primary objective of Pseudo-Noise Tuning (PNT) [48] is to bolster the stability of model training and enhance the performance of DNN models post-quantization. This enhancement is achieved by introducing Pseudo-Quantization Noise (PQN) during the model's fine-tuning phase. The strategic integration of PQN allows the model to more effectively adapt to and anticipate errors stemming

Table 10: Network Topology. In the network topology descriptions, 'C' represents a convolutional layer, 'A' stands for an activation layer (ReLU), 'P' denotes a pooling layer (average pooling), and 'F' indicates a fully connected layer. The sequence and grouping of these letters describe the arrangement of layers in each model. For instance, "[C-A-C-A-P] x 2" means two consecutive blocks each containing two convolutional layers with ReLU activation followed by a pooling layer.

Dataset	Model Name	Network Topology
CIFAR-10	VGG9	[C-A-C-A-P] x 2 - [C-A-C-A-P] x 1 - F-F-F
CIFAR-10	VGG6-96	[C-A-P] - [C-A-P] - [C-A] - F - F - F
CIFAR-10	VGG6-32	[C-A-P] - [C-A-P] - [C-A] - F - F - F
CIFAR-10	VGG6-24	[C-A-P] - [C-A-P] - [C-A] - F - F - F
FashionMNIST	Net-128	F-F
MNIST	Net-30	F-F

Table 11: Input/Output Channels. In the input/output channel specifications, the format "xx/yy" indicates the number of input and output channels for each layer or block of layers. 'xx' represents the number of input features to the layer, and 'yy' denotes the number of output features produced by the layer. This format helps in understanding the dimensionality transformation at each stage of the network.

Dataset	Model Name	Input/Output Channels
CIFAR-10	VGG9	[3/64, 64/64, 64/128, 128/128,128/256, 256/256] - [4096/512, 512/256, 256/10]
CIFAR-10	VGG6-96	[3/96, 96/96, 96/64] - [4096/2048, 2048/1024, 1024/10]
CIFAR-10	VGG6-32	[3/32, 32/32, 32/32] - [2048/1024,1024/512, 512/10]
CIFAR-10	VGG6-24	[3/24, 24/24, 24/24] - [1536/768, 768/384, 384/10]
FashionMNIST	Net-128	[784/128, 128/10]
MNIST	Net-30	[784/30, 30/10]

Table 12: Compare the computation times of linear computations and activation function. This experiment performs encrypted inference on an image from the CIFAR-10 dataset. We use a computer equipped with an AMD Ryzen Threadripper PRO 5975WX processor , 256 GB of RAM and Four NVIDIA RTX 4090 GPUs.

Model	ReLU	CNN/FC	Total
VGG9	83 s	2 s	85 s
VGG6-96	47 s	2 s	49 s
VGG6-32	15 s	2 s	17 s
VGG6-24	12 s	<1 s	13 s

from the quantization process, thus optimizing its overall effectiveness.

The noise proxy function, $f_{Noise}(x \mid \gamma, \Delta, e)$, plays an integral role in the Pseudo-Noise Tuning. This function is tasked with quantizing network parameters, where it determines the quantized output based on the input parameter x, the truncation boundary γ, and the bit width θ. In this context, e denotes the Pseudo-Quantization Noise that influences the quantization process. The quantization step, Δ, is defined using the formula: $\Delta = \frac{\gamma}{2^{\theta-1}-1}$, as visualized in Figure 9 which shows the 2-bit quantization with hyper parameters for symmetric distribution. The rules for quantization are described as follows:

$$f_{Noise}(x \mid \gamma, \Delta, e) = \begin{cases} 0, & \text{if } x \leq 0 \\ x + e \cdot \Delta, & \text{if } 0 < x < \gamma \\ \gamma, & \text{if } x \geq \gamma \end{cases} \tag{7}$$

Algorithm 4 details the procedure for Pseudo-Noise Tuning. The inputs include training data $\mathbf{X} \in \mathbb{R}^{h \times m}$, label matrix $\mathbf{T} \in \mathbb{R}^{q \times m}$, pre-trained model weights $\mathbf{W}1, \mathbf{W}2, \ldots, \mathbf{W}n$, truncation boundaries $\gamma_1 \sim \gamma_n$, quantization steps $\Delta_1 \sim \Delta_n$, along with the noise proxy function f_{Noise} and the pseudo-noise distribution $\mathcal{D}_{Noise}$. This fine-tuning method involves adding noise using the proxy function f_{Noise} after each ReLU layer during the forward pass. The rest of the process follows the same steps as standard fine-tuning as described in references [50] [7]. The final output of Pseudo-Noise Tuning is a model with weights adapted to the noise characteristics of the $\mathcal{D}_{Noise}$ distribution.

In the FHEW Approx-DNN inference process, the accuracy losses from quantized models and activation values increase due to the noise from the FHEW Approx-Neuron. To address these challenges, we use a method similar to Pseudo-Noise Tuning, which significantly enhances the model's resilience to the typical noise of the FHEW Approx-Neuron. We elaborate on the methodology and its effects in § 4.2.

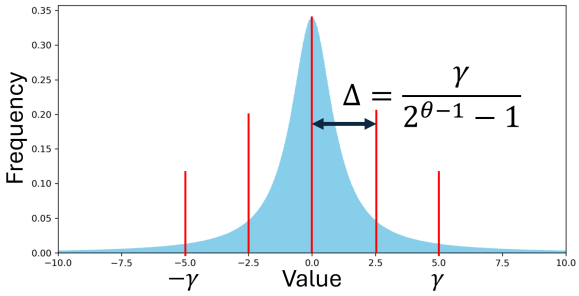

Figure 9: Visualization of the 2-bit quantization with hyper parameters (e.g., γ for truncation boundary and θ for bit-width) for symmetric distribution.

Algorithm 4 Pseudo-Noise Tuning (PNT)

Inputs:
- Training data $\mathbf{X} \in \mathbb{R}^{h \times m}$, where h is the number of features per input sample, and m is the batch size, representing the total number of samples.
- The label matrix $\mathbf{T} \in \mathbb{R}^{q \times m}$ where q is the number of classes for multi-class classification and m matches the batch size in $\mathbf{X}$.
- Pre-trained model weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$ with n layers,$\mathbf{W_i} \in \mathbb{R}^{h \times h}$, where k is the number of output features and h is the number of input features per sample.
- Truncation Boundary of each layer $\gamma_1 \sim \gamma_n$.
- Quantization Step of each layer $\Delta_1 \sim \Delta_n$.
- Pseudo-Noise Distribution $\mathcal{D}_{Noise}$.
- Noise Proxy Function f_{Noise}.

Outputs: Fine-Tuned model weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$ with n layers, $\mathbf{W}_i \in \mathbb{R}^{k \times h}$.

procedure PNT($\mathbf{X}, \mathbf{T}, \mathbf{W}_1 \sim \mathbf{W}_n, \gamma_1 \sim \gamma_n, \Delta_1 \sim \Delta_n, \mathcal{D}_{Noise}$)$\rightarrow$ $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$
  Initialize the input $\mathbf{Y}_0 = \mathbf{X}$.
  for $i = 1$ to n **do**
    $\mathbf{Y}_i = \mathbf{W}_i \cdot \mathbf{Y}_{i-1}$
    $\mathbf{Y}_i = \text{ReLU}(\mathbf{Y}_i)$
    ▷ Start Noise Proxy /////
    $e \sim \mathcal{D}_{Noise}$
    $\mathbf{Y}_i \leftarrow f_{Noise}(\mathbf{Y}_i \mid \gamma_i, \Delta_i, e)$
    ▷ End Noise Proxy /////
  end for
  Compute loss $L(\mathbf{Y}_n, \mathbf{T})$ using Cross-Entropy Loss
  Update weights $\mathbf{W}_1, \ldots, \mathbf{W}_n$ using backpropagation with SGD
  Return $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_n$
end procedure